Contents

Chapter 1

Alexander Bogomolny's UnOrdered Permutation Algorithm

Alexander Bogomolny's UnOrdered Permutation Algorithm - GeeksforGeeks

The Alexander Bogomolyn's algorithm is used to permute first N natural numbers. Given the value of N we have to output all the permutations of numbers from 1 to N.

Examples:

```
Input : 2
Output : 1 2
         2 1

Input : 3
Output : 1 2 3
         1 3 2
         2 1 3
         3 1 2
         2 3 1
         3 2 1
```

The idea is to maintain an array to store the current permutation. A static integer level variable is used to define these permutations.

1. It initializes the value of current level and permutes the remaining values to the higher levels.
2. As the assigning action of the values reaches to the highest level, it prints the permutation obtained.

3. This approach is recursively implemented to obtain all possible permutations.

C++

```
 // CPP program to implement Alexander
// Bogomolny's UnOrdered Permutation Algorithm
#include <iostream>
using namespace std;

// A function to print the permutation.
void print(int perm[], int N)
{
    for (int i = 0; i < N; i++)
        cout << " " << perm[i];
    cout << "\n";
}

// A function implementing Alexander Bogomolyn
// algorithm.
void AlexanderBogomolyn(int perm[], int N, int k)
{
    static int level = -1;

    // Assign level to zero at start.
    level = level + 1;
    perm[k] = level;

    if (level == N)
        print(perm, N);
    else
        for (int i = 0; i < N; i++)

            // Assign values to the array
            // if it is zero.
            if (perm[i] == 0)
                AlexanderBogomolyn(perm, N, i);

    // Decrement the level after all possible
    // permutation after that level.
    level = level - 1;

    perm[k] = 0;
}

// Driver Function
int main()
{
    int i, N = 3;
    int perm[N] = { 0 };
    AlexanderBogomolyn(perm, N, 0);
    return 0;
}
```

Java

```
 // Java program to implement
// Alexander Bogomolny UnOrdered
// Permutation Algorithm
import java.io.*;

class GFG
{
static int level = -1;

// A function to print
// the permutation.
static void print(int perm[], int N)
{
    for (int i = 0; i < N; i++)
        System.out.print(" " + perm[i]);
    System.out.println();
}

// A function implementing
// Alexander Bogomolyn algorithm.
static void AlexanderBogomolyn(int perm[],
                               int N, int k)
{

    // Assign level to
    // zero at start.
    level = level + 1;
    perm[k] = level;

    if (level == N)
        print(perm, N);
    else
        for (int i = 0; i < N; i++)

            // Assign values
            // to the array
            // if it is zero.
            if (perm[i] == 0)
                AlexanderBogomolyn(perm, N, i);

    // Decrement the level
    // after all possible
    // permutation after
    // that level.
    level = level - 1;

    perm[k] = 0;
}

// Driver Code
public static void main (String[] args)
{
    int i, N = 3;
    int perm[] = new int[N];
    AlexanderBogomolyn(perm, N, 0);
}
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to implement
// Alexander Bogomolny UnOrdered
// Permutation Algorithm
using System;

class GFG
{
static int level = -1;

// A function to print
// the permutation.
static void print(int []perm,
                  int N)
{
    for (int i = 0; i < N; i++)
        Console.Write(" " + perm[i]);
    Console.WriteLine();
}

// A function implementing
// Alexander Bogomolyn algorithm.
static void AlexanderBogomolyn(int []perm,
                               int N, int k)
{

    // Assign level to
    // zero at start.
    level = level + 1;
    perm[k] = level;

    if (level == N)
        print(perm, N);
    else
        for (int i = 0; i < N; i++)

            // Assign values
            // to the array
            // if it is zero.
            if (perm[i] == 0)
                AlexanderBogomolyn(perm, N, i);

    // Decrement the level
    // after all possible
    // permutation after
    // that level.
    level = level - 1;

    perm[k] = 0;
}

// Driver Code
public static void Main ()
{
    int N = 3;
    int []perm = new int[N];
    AlexanderBogomolyn(perm, N, 0);
}
}

// This code is contributed
// by anuj_67.
```

Output:

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

Improved By : vt_m

Source

https://www.geeksforgeeks.org/alexander-bogomolnys-unordered-permutation-algorithm/

Chapter 2

All permutations of a string using iteration

All permutations of a string using iteration - GeeksforGeeks

A permutation, also called an “arrangement number” or “order”, is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! permutation (Source: Mathword)

Below are the permutations of string ABC.
ABC ACB BAC BCA CBA CAB

We have discussed different recursive approaches to print permutations here and here.

How to print all permutations iteratively?
A **simple solution** to use permutations of n-1 elements to generate permutations of n elements.

For example let us see how to generate permutations of size 3 using permutations of size 2.

Permutations of two elements are 1 2 and 2 1.
Permutations of three elements can be obtained by inserting 3 at different positions in all permutations of size 2.
Inserting 3 in different positions of 1 2 leads to 1 2 3, 1 3 2 and 3 1 2.
Inserting 3 in different positions of 2 1 leads to 2 1 3, 2 3 1 and 3 2 1.

To generate permutations of size four, we consider all above six permutations of size three and insert 4 at different positions in every permutation.

An efficient solution is to use Johnson and Trotter algorithm to generate all permutations iteratively.

Source

https://www.geeksforgeeks.org/permutations-string-using-iteration/

Chapter 3

All possible strings of any length that can be formed from a given string

All possible strings of any length that can be formed from a given string - GeeksforGeeks

Given a string of distinct characters, print all possible strings of any length that can be formed from given string characters.

Examples:

```
Input  : abc
Output : a b c abc ab ac bc bac bca
         cb ca ba cab cba acb

Input : abcd
Output : a b ab ba c ac ca bc cb abc acb bac
         bca cab cba d ad da bd db abd adb bad
         bda dab dba cd dc acd adc cad cda dac
         dca bcd bdc cbd cdb dbc dcb abcd abdc
         acbd acdb adbc adcb bacd badc bcad bcda
         bdac bdca cabd cadb cbad cbda cdab cdba
         dabc dacb dbac dbca dcab dcba
```

The generation of all strings include following steps.
1) Generate all subsequences of given string.
2) For every subsequence 'subs', print all permutations of 'subs'

Below is C++ implementation. It uses next_permutation function in C++.

```
 /*  C++ code to generate all possible strings
```

```
    that can be formed from given string */
#include<bits/stdc++.h>
using namespace std;

void printAll(string str)
{
    /* Number of subsequences is (2**n -1)*/
    int n = str.length();
    unsigned int opsize = pow(2, n);

    /* Generate all subsequences of given strint.
       using counter 000..1 to 111..1*/
    for (int counter = 1; counter < opsize; counter++)
    {
        string subs = "";
        for (int j = 0; j < n; j++)
        {
            /* Check if jth bit in the counter is set
                If set then print jth element from arr[] */
            if (counter & (1<<j))
                subs.push_back(str[j]);
        }

        /* Print all permutations of current subsequence */
        do
        {
            cout << subs << " ";
        }
        while (next_permutation(subs.begin(), subs.end()));
    }
}

// Driver program
int main()
{
    string str = "abc";
    printSubsequences(str);
    return 0;
}
```

Output:

```
a b ab ba c ac ca bc cb abc acb bac bca cab cba
```

Source

https://www.geeksforgeeks.org/possible-strings-length-can-formed-given-string/

Chapter 4

C program to calculate the value of nPr

C program to calculate the value of nPr - GeeksforGeeks

nPr represents n permutationr and value of nPr is (n!) / (n-r)!.

```
 #include<stdio.h>

int fact(int n)
{
    if (n <= 1)
        return 1;
    return n*fact(n-1);
}

int nPr(int n, int r)
{
    return fact(n)/fact(n-r);
}

int main()
{
    int n, r;
    printf("Enter n: ");
    scanf("%d", &n);

    printf("Enter r: ");
    scanf("%d", &r);

    printf("%dP%d is %d", n, r, nPr(n, r));

    return 0;
}
```

```
Enter n: 5
Enter r: 2
5P2 is 20
```

Please refer Permutation Coefficient for efficient methods to compute nPr.

Source

https://www.geeksforgeeks.org/c-program-calculate-value-npr/

Chapter 5

Cartesian Product of Two Sets

Cartesian Product of Two Sets - GeeksforGeeks

Let A and B be two sets, Cartesian product**A × B** is the set of all ordered pair of elements from A and B
A × B = {{x, y} : x A, y B}

> Let A = {a, b, c} and B = {d, e, f}
> The Cartesian product of two sets is
> A x B = {a, d}, {a, e}, {a, f}, {b, d}, {b, e}, {b, f}, {c, d}, {c, e}, {c, f}}
>
> A has 3 elements and B also has 3 elements. The Cartesian Product has 3 x 3 = 9 elements.
>
> In general, if there are m elements in set A and n elements in B, the number of elements in the Cartesian Product is **m x n**

Given two finite non-empty sets, write a program to print Cartesian Product.

Examples :

```
Input : A = {1, 2}, B = {3, 4}
Output : A × B = {{1, 3}, {1, 4}, {2, 3}, {2, 4}}

Input  : A = {1, 2, 3} B = {4, 5, 6}
Output : A × B = {{1, 4}, {1, 5}, {1, 6}, {2, 4},
         {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}}
```

CPP

```
 // C++ Program to find the Cartesian Product of Two Sets
#include <stdio.h>
```

```
void findCart(int arr1[], int arr2[], int n, int n1)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n1; j++)
            printf("{%d, %d}, ", arr1[i], arr2[j]);
}

int main()
{
    int arr1[] = { 1, 2, 3 }; // first set
    int arr2[] = { 4, 5, 6 }; // second set
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    findCart(arr1, arr2, n1, n2);
    return 0;
}
```

Java

```
 // Java Program to find the
// Cartesian Product of Two Sets
import java.io.*;
import java.util.*;

class GFG {

    static void findCart(int arr1[], int arr2[],
                                int n, int n1)
    {
        for (int i = 0; i < n; i++)
          for (int j = 0; j < n1; j++)
            System.out.print("{"+ arr1[i]+", "
                             + arr2[j]+"}, ");
    }
    // Driver code
    public static void main (String[] args) {

        // first set
        int arr1[] = { 1, 2, 3 };

        // second set
        int arr2[] = { 4, 5, 6 };

        int n1 = arr1.length;
        int n2 = arr2.length;
        findCart(arr1, arr2, n1, n2);
    }
}


// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python3 Program to find the
# Cartesian Product of Two Sets

def findCart(arr1, arr2, n, n1):

    for i in range(0,n):
        for j in range(0,n1):
            print("{",arr1[i],", ",arr2[j],"}, ",sep="",end="")

# Driver code
arr1 = [ 1, 2, 3 ] # first set
arr2 = [ 4, 5, 6 ] # second set

n1 = len(arr1) # sizeof(arr1[0])
n2 = len(arr2) # sizeof(arr2[0]);

findCart(arr1, arr2, n1, n2);

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# Program to find the
// Cartesian Product of Two Sets
using System;

class GFG {

    static void findCart(int []arr1, int []arr2,
                                    int n, int n1)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n1; j++)
                Console.Write("{" + arr1[i] + ", "
                                + arr2[j] + "}, ");
    }

    // Driver code
    public static void Main () {
        // first set
        int []arr1 = { 1, 2, 3 };

        // second set
        int []arr2 = { 4, 5, 6 };

        int n1 = arr1.Length;
        int n2 = arr2.Length;

        findCart(arr1, arr2, n1, n2);
    }
}


// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the
// Cartesian Product of Two Sets

function findCart($arr1, $arr2, $n, $n1)
{
    for ($i = 0; $i < $n; $i++)
        for ( $j = 0; $j < $n1; $j++)
            echo "{", $arr1[$i] ," , ",
                      $arr2[$j], "}",",";
}

// Driver Code

// first set
$arr1 = array ( 1, 2, 3 );

// second set
$arr2 = array ( 4, 5, 6 );
$n1 = sizeof($arr1) ;
$n2 = sizeof($arr2);
findCart($arr1, $arr2, $n1, $n2);

// This code is contributed by m_kit.
?>
```

Output :

```
{{1, 4}, {1, 5}, {1, 6}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}}
```

Practical Examples:
1) A set of playing cards is Cartesian product of a four element set to a set of 13 elements.

2) A two dimensional coordinate system is a Cartesian product of two sets of real numbers.

Reference:
https://en.wikipedia.org/wiki/Cartesian_product

Improved By : jit_t

Source

https://www.geeksforgeeks.org/cartesian-product-two-sets/

Chapter 6

Check if a binary string contains all permutations of length k

Check if a binary string contains all permutations of length k - GeeksforGeeks

Given a binary string and k, to check whether it's contains all permutations of length k or not.

Examples:

```
Input : Binary string 11001
        k : 2
Output : Yes
11001 contains all possibilities of
binary sequences with k = 2, 00, 01,
10, 11

Input : Binary string: 1001
        k : 2
Output: No
1001 does not contain all possibilities of
binary sequences with k = 2. Here 11
sequence is missing
```

Explanation:
In this example one binary sequence of length k is not found it is 0110.

So all binary sequences with k=4 will be 2^4=16. they are following
0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
All should be sub string of given binary string then print Yes otherwise No

Algorithm
Taking binary string and the size k. In binary string we check binary sequences are matched

or not. Binary sequence is made of size k, as we know that in binary using 0 and 1 digit so to generate total binary subsets is 2^k element. The main idea behind it, to store all binary value in list as string and then compare list all item to given binary string as subset. If all are occur inside the binary string then print “Yes” otherwise print “No”.

JAVA

```
 // Java program to Check a binary string
// contains all permutations of length k.

import java.util.*;
public class Checkbinary {

    // to check all Permutation in given String
    public static boolean tocheck(String s, int k)
    {
        List<String> list = BinarySubsets(k);

        // to check binary sequences are available
        // in string or not
        for (String b : list)
            if (s.indexOf(b) == -1)
                return false;

        return true;
    }

    // to generate all binary subsets of given length k
    public static List<String> BinarySubsets(int k)
    {
        // Decalre the list as String
        List<String> list = new ArrayList<>();

        // to define the format of binary of
        // given length k
        String format = "%0" + k + "d";

        // returns the string representation of the
        // unsigned integer value represented by
        // the argument in binary (base 2)  using
        // Integer.toBinaryString and convert it
        // into integer using Integer.valueOf.
        // Loop for 2<sup>k</sup> elements
        for (int i = 0; i < Math.pow(2, k); i++)
        {
            // To add in the list all possible
            // binary sequence of given length
            list.add(String.format(format,
                Integer.valueOf(Integer.toBinaryString(i))));
            /* To Show all binary sequence of given
               length k
            System.out.println(String.format(format,
            Integer.valueOf(Integer.toBinaryString(i))));*/
        }
        return list;
    }

    // drive main
    public static void main(String[] args)
    {
        String str = "11001";
        int num = 2;
        if (tocheck(str, num))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

Output:

```
Yes
```

Source

https://www.geeksforgeeks.org/check-if-a-binary-string-contains-all-permutations-of-length-k/

Chapter 7

Combination and Permutation Practice Questions | Set 1

Combination and Permutation Practice Questions | Set 1 - GeeksforGeeks

Prerequisite : Permutation and Combination

n students appear in an examination, find the number of ways the result of examination can be announced.

> Answer is $\mathbf{2^n}$
>
> Examples:
> Input : n = 6
> Output : Each student can either pass or fail in the examination. so ,there exists
> 2
> possibilities for each of the 6 students in the result. hence total number of ways for the result=$(2)^6$
>
> Input : n = 8
> Output :$(2)^8$=256

'n' matches are to be played in class a chess tournament, find the number of ways in which their results can be decided

> Answer is $(3)^n$ ways
>
> Examples:
> Input : n = 3
> Output: The results of each of the 3 matches can be three ways namely win ,draw or loss
> since total no. of ways in which results of 3 matches can be decided =$\mathbf{(3)^3}$
>
> Input: **6**
> Output:$\mathbf{(3)^4}$**=81**

A badminton tournament consists of ‘n’ matches.
(i) Find the number of ways in which their results can be forecast are given.
(ii) Total number of forecasts containing all correct results.
(ii) Total number of forecasts containing all wrong results.

Answer (i) (2^n)
(ii) 1
(iii) 1
Examples:
Input : A badminton tournament consists of 3 matches.
(i) In how many ways can their results be forecast ?
(ii) How many different forecasts can contain all correct results ?
(iii) How many different forecasts can contain all correct results ?
Output:(i) Each badminton match can be decided in only 2 ways either win or loss for a particular team so total number of ways results of 3 matches can be forecast=2^3=8
(ii) Results of each match can be forecast wrong in only 1 way
Total no. of forecasts containing all wrong results = (1^3) = 1
(iii) Similarly, result of each can be forecast correct in only 1 way.
total no .of forecasts containing all correct results = (1^3) = 1

Find the number of ways in which ‘n’ different beads can be arranged to form a necklace

Answer is (n-1)!/2
Examples: For example 4 beads can be arranged in following ways.
….b1

b2…….b4

….b3

….b1

b3…….b2

….b4

….b1

b4…….b3

….b2

Since it does not matter where we place first bead. Total ways to arrange is (n − 1)!. But clockwise and anticlockwise arrangements are same, so total arrangements are (n − 1)!/2

There are ‘n’ questions papers, find the no, of ways in which a student can attempt one or more questions

Answer: **(2^n-1)ways.**

For example a student will solve one or more questions out of 4 questions in following ways.
1) The student chooses to solve only one question, can choose in $^4C_1$
2) The student chooses to solve only two questions, can choose in $^4C_2$
3) The student chooses to solve only three questions, can choose in $^4C_3$
3) The student chooses to solve all four questions, can choose in $^4C_4$
So total ways are

$^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4$
$=2^4$-1 = 15 ways
We know sum of binomial coefficients from $^nC_0$ to ^nC_n is 2^n

More practice questions on permutation and combination :
Quiz on Permutation and Combination
Combination and Permutation Practice Questions

Source

https://www.geeksforgeeks.org/combination-permutation-practice-questions/

Chapter 8

Combinational Sum

Combinational Sum - GeeksforGeeks

Given an array of positive integers **arr**[] and a sum **x**, find all unique combinations in arr[] where the sum is equal to x. The same repeated number may be chosen from arr[] unlimited number of times. Elements in a combination (a1, a2, …, ak) must be printed in non-descending order. (ie, a1 <= a2 <= … <= ak).
The combinations themselves must be sorted in ascending order, i.e., the combination with smallest first element should be printed first. If there is no combination possible the print "Empty" (without quotes).

Examples:

```
Input : arr[] = 2, 4, 6, 8
            x = 8
Output : [2, 2, 2, 2]
         [2, 2, 4]
         [2, 6]
         [4, 4]
         [8]
```

Since the problem is to get all the possible results, not the best or the number of result, thus we don't need to consider DP(dynamic programming), recursion is needed to handle it.

We should use the following algorithm.

```
1. Sort the array(non-decreasing).
2. First remove all the duplicates from array.
3. Then use recursion and backtracking to solve
   the problem.
   (A) If at any time sub-problem sum == 0 then
       add that array to the result (vector of
       vectors).
   (B) Else if sum if negative then ignore that
       sub-problem.
   (C) Else insert the present array in that
       index to the current vector and call
       the function with sum = sum-ar[index] and
       index = index, then pop that element from
       current index (backtrack) and call the
       function with sum = sum and index = index+1
```

Below is C++ implementation of above steps.

```
 // C++ program to find all combinations that
// sum to a given value
#include <bits/stdc++.h>
using namespace std;

// Print all members of ar[] that have given
void findNumbers(vector<int>& ar, int sum,
                 vector<vector<int> >& res,
                 vector<int>& r, int i)
{
    // If  current sum becomes negative
    if (sum < 0)
        return;

    // if we get exact answer
    if (sum == 0)
    {
        res.push_back(r);
        return;
    }

    // Recur for all remaining elements that
    // have value smaller than sum.
    while (i < ar.size() && sum - ar[i] >= 0)
    {

        // Till every element in the array starting
        // from i which can contribute to the sum
        r.push_back(ar[i]); // add them to list

        // recur for next numbers
        findNumbers(ar, sum - ar[i], res, r, i);
        i++;

        // remove number from list (backtracking)
        r.pop_back();
    }
}

// Returns all combinations of ar[] that have given
// sum.
vector<vector<int> > combinationSum(vector<int>& ar,
                                            int sum)
{
    // sort input array
    sort(ar.begin(), ar.end());

    // remove duplicates
    ar.erase(unique(ar.begin(), ar.end()), ar.end());

    vector<int> r;
    vector<vector<int> > res;
    findNumbers(ar, sum, res, r, 0);

    return res;
}

// Driver code
int main()
{
    vector<int> ar;
    ar.push_back(2);
    ar.push_back(4);
    ar.push_back(6);
    ar.push_back(8);
    int n = ar.size();

    int sum = 8; // set value of sum
    vector<vector<int> > res = combinationSum(ar, sum);

    // If result is empty, then
    if (res.size() == 0)
    {
        cout << "Emptyn";
        return 0;
    }

    // Print all combinations stored in res.
    for (int i = 0; i < res.size(); i++)
    {
        if (res[i].size() > 0)
        {
            cout << " ( ";
            for (int j = 0; j < res[i].size(); j++)
```

```
                cout << res[i][j] << " ";
            cout << ")";
        }
    }
}
```

Output:

```
( 2 2 2 2 ) ( 2 2 4 ) ( 2 6 ) ( 4 4 ) ( 8 )
```

Source

https://www.geeksforgeeks.org/combinational-sum/

Chapter 9

Combinations from n arrays picking one element from each array

Combinations from n arrays picking one element from each array - GeeksforGeeks

Given a list of arrays, find all combinations where each combination contains one element from each given array.

Examples:

```
Input : [ [1, 2], [3, 4] ]
Output : 1 3
         1 4
         2 3
         2 4

Input : [ [1], [2, 3, 4], [5] ]
Output : 1 2 5
         1 3 5
         1 4 5
```

We keep an array of size equal to total no of arrays. This array called indices helps us keep track of the index of current element in each of the n arrays. Initially it is initialized with all 0s indicating current index in each array is that of first element. We keep printing the combinations until no new combinations can be found. Starting from the rightmost array we check if more elements are there in that array. If yes, we increment the entry for that array in indices i.e. move to the next element in that array. We also make the current indices 0 in all the arrays to the right of this array. We keep moving left to check all arrays until one such array is found. If no more arrays are found we stop there.

```
 // CPP program to find combinations from n
// arrays such that one element from each
// array is present
#include <bits/stdc++.h>

using namespace std;

// function to print combinations that contain
// one element from each of the given arrays
void print(vector<vector<int> >& arr)
{
    // number of arrays
    int n = arr.size();

    // to keep track of next element in each of
    // the n arrays
    int* indices = new int[n];

    // initialize with first element's index
    for (int i = 0; i < n; i++)
        indices[i] = 0;

    while (1) {

        // print current combination
        for (int i = 0; i < n; i++)
            cout << arr[i][indices[i]] << " ";
        cout << endl;

        // find the rightmost array that has more
        // elements left after the current element
        // in that array
        int next = n - 1;
        while (next >= 0 &&
              (indices[next] + 1 >= arr[next].size()))
            next--;

        // no such array is found so no more
        // combinations left
        if (next < 0)
            return;

        // if found move to next element in that
        // array
        indices[next]++;

        // for all arrays to the right of this
        // array current index again points to
        // first element
        for (int i = next + 1; i < n; i++)
            indices[i] = 0;
    }
}

// driver function to test above function
int main()
{
    // initializing a vector with 3 empty vectors
    vector<vector<int> > arr(3, vector<int>(0, 0));

    // now entering data
    // [[1, 2, 3], [4], [5, 6]]
    arr[0].push_back(1);
    arr[0].push_back(2);
    arr[0].push_back(3);
    arr[1].push_back(4);
    arr[2].push_back(5);
    arr[2].push_back(6);

    print(arr);
}
```

Output:

```
1 4 5
1 4 6
2 4 5
2 4 6
3 4 5
3 4 6
```

Improved By : NimeshDesai

Source

https://www.geeksforgeeks.org/combinations-from-n-arrays-picking-one-element-from-each-array/

Chapter 10

Combinations in a String of Digits

Combinations in a String of Digits - GeeksforGeeks

Given an input string of numbers, find all combinations of numbers that can be formed using digits in the same order.

Examples:

```
Input : 123
Output :1 2 3
        1 23
        12 3
        123

Input : 1234
Output : 1 2 3 4
        1 2 34
        1 23 4
        1 234
        12 3 4
        12 34
        123 4
        1234
```

The problem can be solved using recursion. We keep track of current index in given input string and length of output string so far. In each call to the function, if there are no digits remaining in the input string print the current output string and return. Otherwise, copy current digit to output. From here make two calls, one considering next digit as part of next number(including a space in output string) and one considering next digit as part of current number(no space included). If there are no digits remaining after current digit the second call to the function is omitted because a trailing space doesn't count as a new combination.

```
 // CPP program to find all combination of numbers
// from a given string of digits
#include <iostream>
#include <cstring>
using namespace std;

// function to print combinations of numbers
// in given input string
void printCombinations(char* input, int index,
                     char* output, int outLength)
{
    // no more digits left in input string
    if (input[index] == '\0')
    {
        // print output string & return
        output[outLength] = '\0';
        cout << output << endl;
        return;
    }

    // place current digit in input string
    output[outLength] = input[index];

    // separate next digit with a space
    output[outLength + 1] = ' ';

    printCombinations(input, index + 1, output,
                                   outLength + 2);

    // if next digit exists make a
    // call without including space
    if(input[index + 1] != '\0')
    printCombinations(input, index + 1, output,
                                    outLength + 1);

}

// driver function to test above function
int main()
{
    char input[] = "1214";
    char *output = new char[100];

    // initialize output with empty string
    output[0] = '\0';

    printCombinations(input, 0, output, 0);
    return 0;
}
```

Output:

```
1 2 1 4
1 2 14
1 21 4
1 214
12 1 4
12 14
121 4
1214
```

Alternative Solution:

```
 // CPP program to find all combination of
// numbers from a given string of digits
// using bit algorithm used same logic
// as to print power set of string
#include <bits/stdc++.h>
using namespace std;

// function to print combinations of
// numbers in given input string
void printCombinations(char s[]){

    // find length of char array
    int l = strlen(s);

    // we can give space between characters
    // ex. ('1' & '2') or ('2' & '3') or
    // ('3' & '4') or ('3' & '4') or all
    // that`s why here we have maximum
    // space length - 1
    for(int i = 0; i < pow(2, l - 1); i++){
        int k = i, x = 0;

        // first character will be printed
        // as well
        cout << s[x];
        x++;
        for(int j = 0; j < strlen(s) - 1; j++){

            // if bit is set, means provide
            // space
            if(k & 1)
```

```
                cout << " ";
            k = k >> 1;
            cout << s[x];

            // always increment index of
            // input string
            x++;
        }
        cout << "\n";
    }
}

// driver code
int main() {

    char input[] = "1214";
    combination(input);

    return 0;
}
// This code is contributed by PRINCE Gupta 2
```

Output:

```
1214
1 214
12 14
1 2 14
121 4
1 21 4
12 1 4
1 2 1 4
```

Source

https://www.geeksforgeeks.org/combinations-string-digits/

Chapter 11

Combinations with repetitions

Combinations with repetitions - GeeksforGeeks

Suppose we have a string of length- n and we want to generate all combinations/permutations taken r at a time with/without repetitions. There are four fundamental concepts in Combinatorics

1) Combinations without repetitions/replacements.
2) Combinations with repetitions/replacements.
3) Permutations without repetitions/replacements.
4) Permutations with repetitions/replacements.

Below is a summary table depicting the fundamental concepts in Combinatorics Theory.

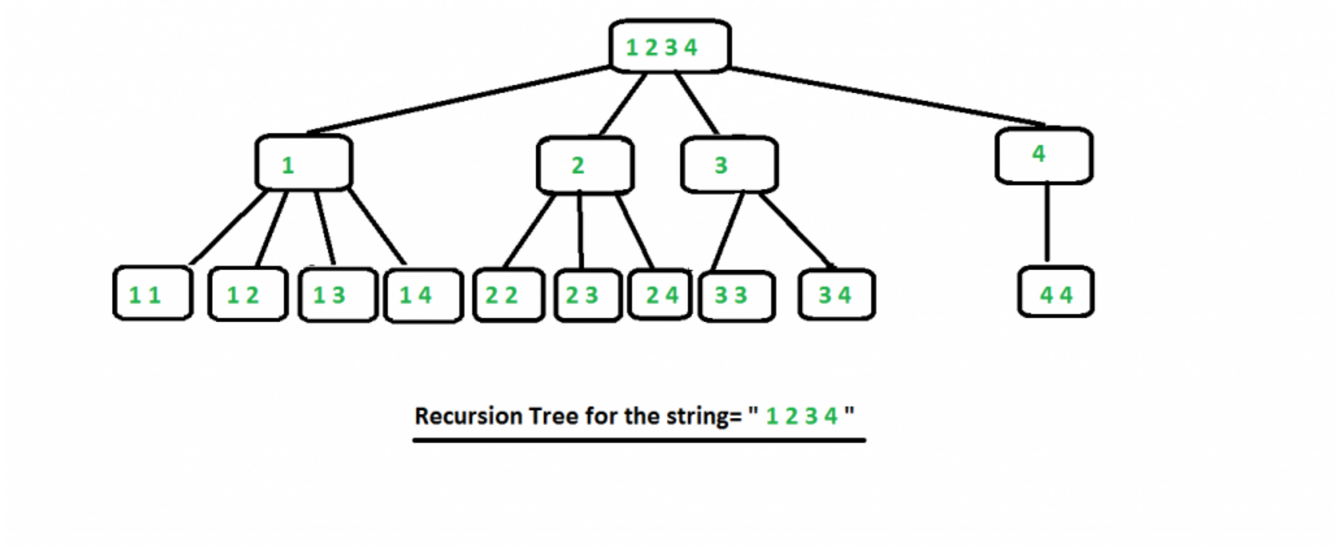
Summary Table

	Replacements/Repetitions allowed	**Replacements/Repetitions not allowe**
Permutations/Order Important	$\mathbf{n^r}$ possibilities https://www.geeksforgeeks.org/print-all-combinations-of-given-length/ See the special case when r=n below https://www.geeksforgeeks.org/print-all-permutations-with-repetition-of-characters	$\mathbf{{}^nP_r}$ possibilities https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutation **Here r=n, as we are permuting all the characters of the string.**
Combinations/Order Not Important	$\mathbf{{}^{n+r-1}C_r}$ possibilities **Current Article (https://www.geeksforgeeks.org/combinations-with-repetitions)**	$\mathbf{{}^nC_r}$ possibilities https://www.geeksforgeeks.org/print-all-possible-combinations-of-r-elemen

This article is about the third case(Order Not important and Repetitions allowed).

The idea is to recur for all the possibilities of the string, even if the characters are repeating.

The base case of the recursion is when there is a total of 'r' characters and the combination is ready to be printed.
For clarity, see the recursion tree for the string- " 1 2 3 4" and r=2



Below is C++ implementation.

```
 // Program to print all combination of size r in an array
// of size n with repetitions allowed
#include <stdio.h>

/* arr[]  ---> Input Array
  chosen[] ---> Temporary array to store indices of
                   current combination
   start & end ---> Staring and Ending indexes in arr[]
   r ---> Size of a combination to be printed */
void CombinationRepetitionUtil(int chosen[], int arr[],
                    int index, int r, int start, int end)
{
    // Since index has become r, current combination is
    // ready to be printed, print
    if (index == r)
    {
        for (int i = 0; i < r; i++)
            printf("%d ", arr[chosen[i]]);
        printf("\n");
        return;
    }
    // One by one choose all elements (without considering
    // the fact whether element is already chosen or not)
    // and recur
    for (int i = start; i <= end; i++)
    {
        chosen[index] = i;
        CombinationRepetitionUtil(chosen, arr, index + 1,
                                               r, i, end);
    }
    return;
}

// The main function that prints all combinations of size r
// in arr[] of size n with repitions. This function mainly
// uses CombinationRepetitionUtil()
void CombinationRepetition(int arr[], int n, int r)
{
    // Allocate memory
    int chosen[r+1];

    // Call the recursice function
    CombinationRepetitionUtil(chosen, arr, 0, r, 0, n-1);
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    int r = 2;
    CombinationRepetition(arr, n, r);
    return 0;
}
```

Output :

```
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

Time Complexity : For a string of length- **n** and combinations taken **r** at a time with repetitions, it takes a total of $\mathbf{O({}^{n+r-1}C_r)}$ time.

References– https://en.wikipedia.org/wiki/Combination

Source

https://www.geeksforgeeks.org/combinations-with-repetitions/

Chapter 12

Combinatorial Game Theory | Set 4 (Sprague – Grundy Theorem)

Combinatorial Game Theory | Set 4 (Sprague - Grundy Theorem) - GeeksforGeeks

Prerequisites : Grundy Numbers/Nimbers and Mex

We have already seen in Set 2 (https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/), that we can find who wins in a game of Nim without actually playing the game.

Suppose we change the classic Nim game a bit. This time each player can only remove 1, 2 or 3 stones only (and not any number of stones as in the classic game of Nim). Can we predict who will win?

Yes, we can predict the winner using Sprague-Grundy Theorem.

What is Sprague-Grundy Theorem?
Suppose there is a composite game (more than one sub-game) made up of N sub-games and two players, A and B. Then Sprague-Grundy Theorem says that if both A and B play optimally (i.e., they don't make any mistakes), then the player starting first is guaranteed to win if the XOR of the grundy numbers of position in each sub-games at the beginning of the game is non-zero. Otherwise, if the XOR evaluates to zero, then player A will lose definitely, no matter what.

How to apply Sprague Grundy Theorem ?
We can apply Sprague-Grundy Theorem in any impartial gameand solve it. The basic steps are listed as follows:

1. Break the composite game into sub-games.
2. Then for each sub-game, calculate the Grundy Number at that position.
3. Then calculate the XOR of all the calculated Grundy Numbers.

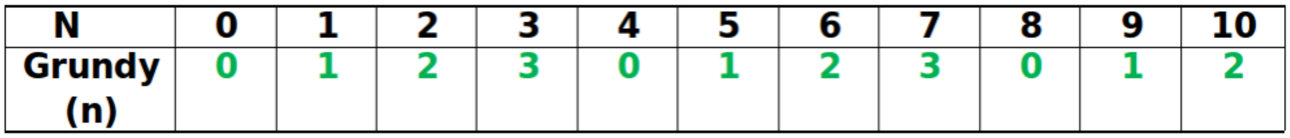
4. If the XOR value is non-zero, then the player who is going to make the turn (First Player) will win else he is destined to lose, no matter what.

Example Game : The game starts with 3 piles having 3, 4 and 5 stones, and the player to move may take any positive number of stones upto 3 only from any of the piles [Provided that the pile has that much amount of stones]. The last player to move wins. Which player wins the game assuming that both players play optimally?

How to tell who will win by applying Sprague-Grundy Theorem?
As, we can see that this game is itself composed of several sub-games.

First Step : The sub-games can be considered as each piles.
Second Step : We see from the below table that

```
Grundy(3) = 3
Grundy(4) = 0
Grundy(5) = 1
```



N	0	1	2	3	4	5	6	7	8	9	10
Grundy (n)	0	1	2	3	0	1	2	3	0	1	2

We have already seen how to calculate the Grundy Numbers of this game in the previous article.

Third Step : The XOR of 3, 4, 5 = 2

Fourth Step : Since XOR is a non-zero number, so we can say that the first player will win.

Below is C++ program that implements above 4 steps.

```
 /*  Game Description-
 "A game is played between two players and there are N piles
 of stones such that each pile has certain number of stones.
 On his/her turn, a player selects a pile and can take any
 non-zero number of stones upto 3 (i.e- 1,2,3)
 The player who cannot move is considered to lose the game
 (i.e., one who take the last stone is the winner).
 Can you find which player wins the game if both players play
 optimally (they don't make any mistake)? "

 A Dynamic Programming approach to calculate Grundy Number
 and Mex and find the Winner using Sprague - Grundy Theorem. */

#include<bits/stdc++.h>
using namespace std;
```

```
/* piles[] -> Array having the initial count of stones/coins
            in each piles before the game has started.
   n       -> Number of piles

   Grundy[] -> Array having the Grundy Number corresponding to
             the initial position of each piles in the game

   The piles[] and Grundy[] are having 0-based indexing*/

#define PLAYER1 1
#define PLAYER2 2

// A Function to calculate Mex of all the values in that set
int calculateMex(unordered_set<int> Set)
{
    int Mex = 0;

    while (Set.find(Mex) != Set.end())
        Mex++;

    return (Mex);
}

// A function to Compute Grundy Number of 'n'
int calculateGrundy(int n, int Grundy[])
{
    Grundy[0] = 0;
    Grundy[1] = 1;
    Grundy[2] = 2;
    Grundy[3] = 3;

    if (Grundy[n] != -1)
        return (Grundy[n]);

    unordered_set<int> Set; // A Hash Table

    for (int i=1; i<=3; i++)
            Set.insert (calculateGrundy (n-i, Grundy));

    // Store the result
    Grundy[n] = calculateMex (Set);

    return (Grundy[n]);
}

// A function to declare the winner of the game
void declareWinner(int whoseTurn, int piles[],
                    int Grundy[], int n)
{
    int xorValue = Grundy[piles[0]];

    for (int i=1; i<=n-1; i++)
        xorValue = xorValue ^ Grundy[piles[i]];

    if (xorValue != 0)
    {
        if (whoseTurn == PLAYER1)
            printf("Player 1 will win\n");
        else
            printf("Player 2 will win\n");
    }
    else
    {
        if (whoseTurn == PLAYER1)
            printf("Player 2 will win\n");
        else
            printf("Player 1 will win\n");
    }

    return;
}


// Driver program to test above functions
int main()
{
    // Test Case 1
    int piles[] = {3, 4, 5};
    int n = sizeof(piles)/sizeof(piles[0]);

    // Find the maximum element
    int maximum = *max_element(piles, piles + n);

    // An array to cache the sub-problems so that
    // re-computation of same sub-problems is avoided
    int Grundy[maximum + 1];
    memset(Grundy, -1, sizeof (Grundy));

    // Calculate Grundy Value of piles[i] and store it
    for (int i=0; i<=n-1; i++)
        calculateGrundy(piles[i], Grundy);

    declareWinner(PLAYER1, piles, Grundy, n);

    /* Test Case 2
    int piles[] = {3, 8, 2};
    int n = sizeof(piles)/sizeof(piles[0]);


    int maximum = *max_element (piles, piles + n);

    // An array to cache the sub-problems so that
    // re-computation of same sub-problems is avoided
    int Grundy [maximum + 1];
    memset(Grundy, -1, sizeof (Grundy));

    // Calculate Grundy Value of piles[i] and store it
    for (int i=0; i<=n-1; i++)
        calculateGrundy(piles[i], Grundy);

    declareWinner(PLAYER2, piles, Grundy, n);   */

    return (0);
}
```

Output :

```
Player 1 will win
```

References :
https://en.wikipedia.org/wiki/Sprague%E2%80%93Grundy_theorem

Exercise to the Readers: Consider the below game.
"A game is played by two players with N integers A1, A2, .., AN. On his/her turn, a player selects an integer, divides it by 2, 3, or 6, and then takes the floor. If the integer becomes 0, it is removed. The last player to move wins. Which player wins the game if both players play optimally?"

Hint : See the example 3 of previous article.

Improved By : vinayb21

Source

https://www.geeksforgeeks.org/combinatorial-game-theory-set-4-sprague-grundy-theorem/

Chapter 13

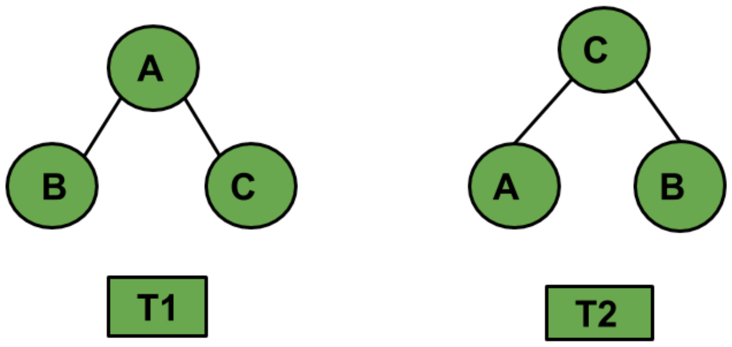
Combinatorics on ordered trees

Combinatorics on ordered trees - GeeksforGeeks

An **ordered tree** is an oriented tree in which the children of a node are somehow ordered. It is a rooted tree in which an ordering is specified for the children of each vertex. This is called a "plane tree" because an ordering of the children is equivalent to an embedding of the tree in the plane, with the root at the top and the children of each vertex lower than that vertex.
Ordered tree can be further specified as labelled ordered tree and unlabelled ordered tree.

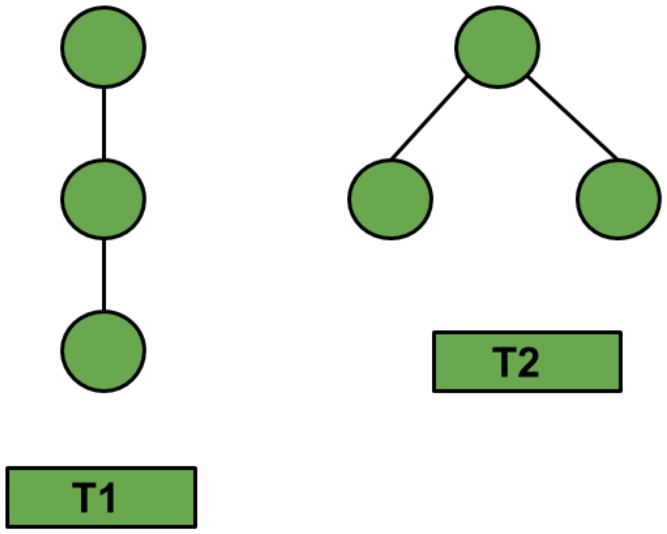
Prerequisite : Catalan Numbers | Binomial Coefficient.

Labelled ordered trees : A labeled tree is a tree where each vertex is assigned a unique number from 1 to n.



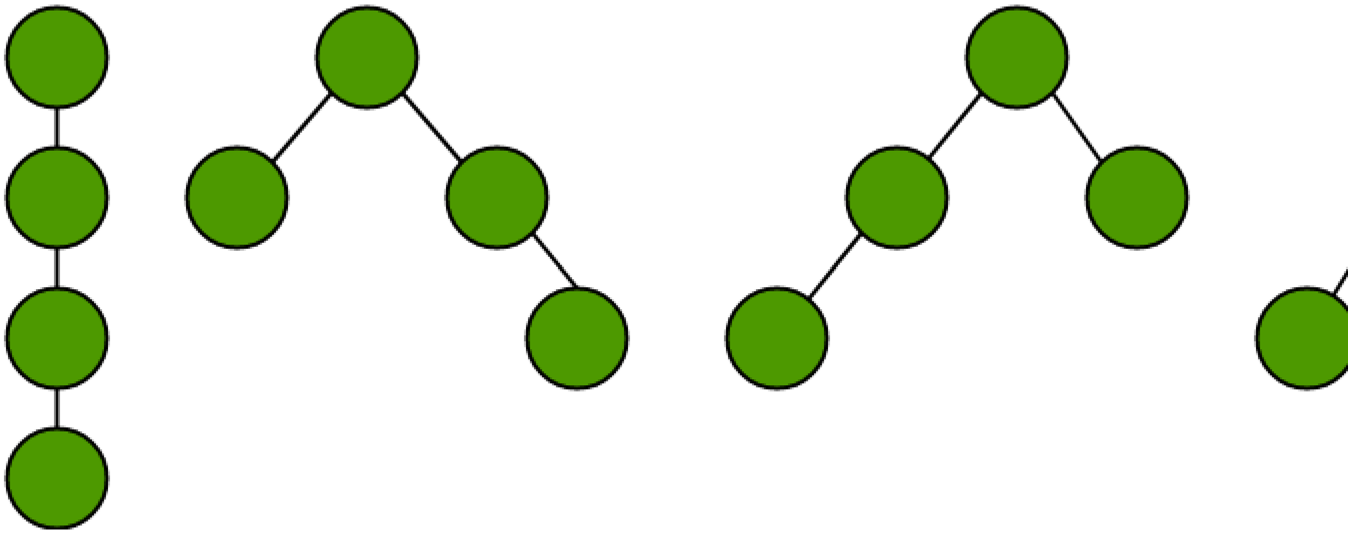
If T1 and T2 are ordered trees. Then, T1 != T2 else T1 = T2.

Unlabelled ordered trees : An unlabelled tree is a tree where every vertex is unlabelled. Given below are the possible unlabelled ordered tree having 3 vertices.



The total number of unlabelled ordered trees having n nodes is equal to the (n − 1)-th Catalan Number.

Given below are the possible unlabelled ordered trees having 4 nodes. This diagram will work as a reference example for the next few results.



1. Number of trees with exactly k leaves.

Let us consider, we have a ‘n’ edges . Then, the solution for the total possible ordered trees having ‘k’ leaves is given by :

$$L_n(k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

2. Total number of nodes of degree d in these trees.

Let us consider, we have a 'n' edges . Then, the solution for the total number of nodes having degree 'd' is given by :

$$D_n(d) = \binom{2n-1-d}{n-1}$$

3. Number of trees in which the root has degree r.

Let us consider, we have a 'n' edges . Then, the solution for the total possible ordered trees whose root has degree 'r' is given by :

$$R_n(r) = \frac{r}{n}\binom{2n-1-r}{n-1}$$

Below is the implementation of above combinatorics functions using Binomial Coefficient :

C++

```
 // CPP code to find the number of ordered trees
// with given number of edges and leaves
#include <bits/stdc++.h>
using namespace std;

// Function returns value of
// Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int C[n + 1][k + 1] = { 0 };
    int i, j;

    // Caculate value of Binomial
    // Coefficient in bottom up manner
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= min(i, k); j++) {

            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using
            // previously stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    return C[n][k];
}

// Function to calculate the number
// of trees with exactly k leaves.
int k_Leaves(int n, int k)
{
    int ans = (binomialCoeff(n, k) * binomialCoeff(n, k - 1)) / n;
    cout << "Number of trees having 4 edges"
         << " and exactly 2 leaves : " << ans << endl;
    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
int numberOfNodes(int n, int d)
{
    int ans = binomialCoeff(2 * n - 1 - d, n - 1);
    cout << "Number of nodes of degree 1 in"
         << " a tree having 4 edges : " << ans << endl;
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
int rootDegreeR(int n, int r)
{
    int ans = r * binomialCoeff(2 * n - 1 - r, n - 1);
    ans = ans / n;
    cout << "Number of trees having 4 edges"
```

```
         << " where root has degree 2 : " << ans << endl;
    return 0;
}

// Driver program to test above functions
int main()
{
    // Number of trees having 3
    // edges and exactly 2 leaves
    k_Leaves(3, 2);

    // Number of nodes of degree
    // 3 in a tree having 4 edges
    numberOfNodes(3, 1);

    // Number of trees having 3
    // edges where root has degree 2
    rootDegreeR(3, 2);

    return 0;
}
```

Java

```
 // java code to find the number of ordered
// trees with given number of edges and
// leaves
import java.io.*;

class GFG {

    // Function returns value of
    // Binomial Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int [][]C = new int[n+1][k+1];
        int i, j;

        // Caculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= Math.min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;
                // Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1]
                               + C[i - 1][j];
            }
        }

        return C[n][k];
    }

    // Function to calculate the number
    // of trees with exactly k leaves.
    static int k_Leaves(int n, int k)
    {
        int ans = (binomialCoeff(n, k) *
                binomialCoeff(n, k - 1)) / n;
        System.out.println( "Number of trees "
             + "having 4 edges and exactly 2 "
                        + "leaves : " + ans) ;
        return 0;
    }

    // Function to calculate total number of
    // nodes of degree d in these trees.
    static int numberOfNodes(int n, int d)
    {
        int ans = binomialCoeff(2 * n - 1 - d,
                                       n - 1);
        System.out.println("Number of nodes "
           +"of degree 1 in a tree having 4 "
                          + "edges : " + ans);
        return 0;
    }

    // Function to calculate the number of
    // trees in which the root has degree r.
    static int rootDegreeR(int n, int r)
    {
        int ans = r * binomialCoeff(2 * n
                              - 1 - r, n - 1);
        ans = ans / n;
        System.out.println("Number of trees "
            + "having 4 edges where root has"
                      + " degree 2 : " + ans);
        return 0;
    }
```

```
    // Driver program to test above functions

    public static void main (String[] args)
    {

        // Number of trees having 3
        // edges and exactly 2 leaves
        k_Leaves(3, 2);

        // Number of nodes of degree
        // 3 in a tree having 4 edges
        numberOfNodes(3, 1);

        // Number of trees having 3
        // edges where root has degree 2
        rootDegreeR(3, 2);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# code to find the number of ordered
// trees with given number of edges and
// leaves
using System;

class GFG {

    // Function returns value of
    // Binomial Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int [,]C = new int[n+1,k+1];
        int i, j;

        // Caculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= Math.Min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i,j] = C[i - 1,j - 1]
                            + C[i - 1,j];
            }
        }

        return C[n,k];
    }

    // Function to calculate the number
    // of trees with exactly k leaves.
    static int k_Leaves(int n, int k)
    {
        int ans = (binomialCoeff(n, k) *
                binomialCoeff(n, k - 1)) / n;
        Console.WriteLine( "Number of trees "
            + "having 4 edges and exactly 2 "
                        + "leaves : " + ans) ;
        return 0;
    }

    // Function to calculate total number of
    // nodes of degree d in these trees.
    static int numberOfNodes(int n, int d)
    {
        int ans = binomialCoeff(2 * n - 1 - d,
                                    n - 1);
        Console.WriteLine("Number of nodes "
        +"of degree 1 in a tree having 4 "
                        + "edges : " + ans);
        return 0;
    }

    // Function to calculate the number of
    // trees in which the root has degree r.
    static int rootDegreeR(int n, int r)
    {
        int ans = r * binomialCoeff(2 * n
                            - 1 - r, n - 1);
        ans = ans / n;
        Console.WriteLine("Number of trees "
            + "having 4 edges where root has"
                    + " degree 2 : " + ans);
        return 0;
    }

    // Driver program to test above functions
```

```
    public static void Main ()
    {

        // Number of trees having 3
        // edges and exactly 2 leaves
        k_Leaves(3, 2);

        // Number of nodes of degree
        // 3 in a tree having 4 edges
        numberOfNodes(3, 1);

        // Number of trees having 3
        // edges where root has degree 2
        rootDegreeR(3, 2);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP code to find the number of ordered
// trees with given number of edges and
// leaves

// Function returns value of Binomial
// Coefficient C(n, k)
function binomialCoeff($n, $k)
{
    $C = array(array());
    $i; $j;

    // Caculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++) {
        for ($j = 0; $j <= min($i, $k); $j++)
        {

            // Base Cases
            if ($j == 0 or $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using
            // previously stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1]
```

```
                              + $C[$i - 1][$j];
        }
    }

    return $C[$n][$k];
}

// Function to calculate the number
// of trees with exactly k leaves.
function k_Leaves( $n, $k)
{
    $ans = (binomialCoeff($n, $k) *
                binomialCoeff($n, $k - 1)) / $n;

    echo "Number of trees having 4 edges and ",
          "exactly 2 leaves : " , $ans ,"\n";

    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
function numberOfNodes( $n, $d)
{
    $ans = binomialCoeff(2 * $n - 1 - $d, $n - 1);
    echo "Number of nodes of degree 1 in"
        , " a tree having 4 edges : " , $ans,"\n" ;
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
function rootDegreeR( $n, $r)
{
    $ans = $r * binomialCoeff(2 * $n - 1 - $r,
                                        $n - 1);
    $ans = $ans / $n;
    echo "Number of trees having 4 edges"
        , " where root has degree 2 : " , $ans ;
    return 0;
}

// Driver program to test above functions
    // Number of trees having 3
    // edges and exactly 2 leaves
    k_Leaves(3, 2);

    // Number of nodes of degree
    // 3 in a tree having 4 edges
    numberOfNodes(3, 1);

    // Number of trees having 3
    // edges where root has degree 2
    rootDegreeR(3, 2);

// This code is contributed by anuj_67.
?>
```

Output:

```
Number of trees having 4 edges and exactly 2 leaves : 3
Number of nodes of degree 1 in a tree having 4 edges : 6
Number of trees having 4 edges where root has degree 2 : 2
```

Time Complexity : O(n*k).
Auxiliary Space : O(n*k).

Improved By : vt_m

Source

https://www.geeksforgeeks.org/combinatorics-ordered-trees/

Chapter 14

Count Derangements (Permutation such that no element appears in its original position)

Count Derangements (Permutation such that no element appears in its original position) - GeeksforGeeks

A Derangement is a permutation of n elements, such that no element appears in its original position. For example, a derangement of {0, 1, 2, 3} is {2, 3, 1, 0}.

Given a number n, find total number of Derangements of a set of n elements.

Examples :

```
Input: n = 2
Output: 1
For two elements say {0, 1}, there is only one
possible derangement {1, 0}

Input: n = 3
Output: 2
For three elements say {0, 1, 2}, there are two
possible derangements {2, 0, 1} and {1, 2, 0}

Input: n = 4
Output: 9
For four elements say {0, 1, 2, 3}, there are 9
possible derangements {1, 0, 3, 2} {1, 2, 3, 0}
```

`{1, 3, 0, 2}, {2, 3, 0, 1}, {2, 0, 3, 1}, {2, 3, 1, 0}, {3, 0, 1, 2}, {3, 2, 0, 1} and {3, 2, 1, 0}`

Let countDer(n) be count of derangements for n elements. Below is recursive relation for it.

```
countDer(n) = (n - 1) * [countDer(n - 1) + countDer(n - 2)]
```

How does above recursive relation work?
There are n – 1 ways for element 0 (this explains multiplication with n – 1).

Let ***0 be placed at index i***. There are now two possibilities, depending on whether or not element i is placed at 0 in return.

1. ***i is placed at 0:*** This case is equivalent to solving the problem for n-2 elements as two elements have just swapped their positions.
2. ***i is not placed at 0:*** This case is equivalent to solving the problem for n-1 elements as now there are n-1 elements, n-1 positions and every element has n-2 choices

Below is Simple Solution based on above recursive formula.

C++

```
 // A Naive Recursive C++ program
// to count derangements
#include <bits/stdc++.h>
using namespace std;

int countDer(int n)
{
// Base cases
if (n == 1) return 0;
if (n == 0) return 1;
if (n == 2) return 1;

// countDer(n) = (n-1)[countDer(n-1) + der(n-2)]
return (n - 1) * (countDer(n - 1) +
                  countDer(n - 2));
}

// Driver Code
int main()
{
    int n = 4;
    cout << "Count of Derangements is "
         << countDer(n);
    return 0;
}
```

Java

```
 // A Naive Recursive java
// program to count derangements
import java.io.*;

class GFG
{

    // Function to count
    // derangements
    static int countDer(int n)
    {
        // Base cases
        if (n == 1) return 0;
        if (n == 0) return 1;
        if (n == 2) return 1;

        // countDer(n) = (n-1)[countDer(n-1) + der(n-2)]
        return (n - 1) * (countDer(n - 1) +
                          countDer(n - 2));
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 4;
        System.out.println( "Count of Derangements is "
                             +countDer(n));

    }
}

// This code is contributed by vt_m
```

Python3

```
 # A Naive Recursive Python3
# program to count derangements

def countDer(n):

    # Base cases
    if (n == 1): return 0
    if (n == 0): return 1
    if (n == 2): return 1
    # countDer(n) = (n-1)[countDer(n-1) + der(n-2)]
    return (n - 1) * (countDer(n - 1) +
                      countDer(n - 2))

# Driver Code
n = 4
print("Count of Derangements is ", countDer(n))


# This code is contributed by Azkia Anam.
```

C#

```
 // A Naive Recursive C#
// program to count derangements
using System;

class GFG
{

    // Function to count
    // derangements
    static int countDer(int n)
    {
        // Base cases
        if (n == 1) return 0;
        if (n == 0) return 1;
        if (n == 2) return 1;

        // countDer(n) = (n-1)[countDer(n-1) + der(n-2)]
        return (n - 1) * (countDer(n - 1) +
                          countDer(n - 2));
    }

    // Driver Code
    public static void Main ()
    {
        int n = 4;
        Console.Write( "Count of Derangements is " +
                        countDer(n));

    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A Naive Recursive PHP program
// to count derangements

function countDer($n)
{

    // Base cases
    if ($n == 1)
        return 0;
    if ($n == 0)
        return 1;
    if ($n == 2)
        return 1;

    // countDer(n) = (n-1)[countDer(n-1) +
    // der(n-2)]
    return ($n - 1) * (countDer($n - 1) +
                       countDer($n - 2));
}

    // Driver Code
    $n = 4;
    echo "Count of Derangements is ", countDer($n);

// This code is contributed by nitin mittal.
?>
```

Output:

```
Count of Derangements is 9
```

Time Complexity: T(n) = T(n-1) + T(n-2) which is exponential.
We can observe that this implementation does repeated work. For example see recursion tree for countDer(5), countDer(3) is being being evaluated twice.

```
cdr() ==> countDer()

                    cdr(5)
                 /         \
             cdr(4)          cdr(3)
           /      \         /     \
       cdr(3)     cdr(2)  cdr(2)   cdr(1)
```

An **Efficient Solution** is to use Dynamic Programming to store results of subproblems in an array and build the array in bottom up manner.

C++

```
 // A Dynamic programming based C++
// program to count derangements
#include <bits/stdc++.h>
using namespace std;

int countDer(int n)
{
    // Create an array to store
    // counts for subproblems
    int der[n + 1];

    // Base cases
    der[0] = 1;
    der[1] = 0;
    der[2] = 1;

    // Fill der[0..n] in bottom up manner
    // using above recursive formula
    for (int i = 3; i <= n; ++i)
        der[i] = (i - 1) * (der[i - 1] +
                            der[i - 2]);

    // Return result for n
    return der[n];
}

// Driver code
int main()
{
    int n = 4;
    cout << "Count of Derangements is "
         << countDer(n);
    return 0;
}
```

Java

```
 // A Dynamic programming based
// java program to count derangements
import java.io.*;

class GFG
{

    // Function to count
    // derangements
    static int countDer(int n)
    {
        // Create an array to store
        // counts for subproblems
        int der[] = new int[n + 1];

        // Base cases
        der[0] = 1;
        der[1] = 0;
        der[2] = 1;

        // Fill der[0..n] in bottom up
        // manner using above recursive
        // formula
        for (int i = 3; i <= n; ++i)
            der[i] = (i - 1) * (der[i - 1] +
                                der[i - 2]);

        // Return result for n
        return der[n];
    }

    // Driver program
    public static void main (String[] args)
    {
        int n = 4;
        System.out.println("Count of Derangements is " +
                            countDer(n));

    }
}

// This code is contributed by vt_m
```

Python3

```
 # A Dynamic programming based Python3
# program to count derangements

def countDer(n):

    # Create an array to store
    # counts for subproblems
    der = [0 for i in range(n + 1)]

    # Base cases
    der[0] = 1
    der[1] = 0
    der[2] = 1

    # Fill der[0..n] in bottom up manner
    # using above recursive formula
    for i in range(3, n + 1):
        der[i] = (i - 1) * (der[i - 1] +
                            der[i - 2])

    # Return result for n
    return der[n]

# Driver Code
n = 4
print("Count of Derangements is ", countDer(n))

# This code is contributed by Azkia Anam.
```

C#

```
// A Dynamic programming based
// C# program to count derangements
using System;

class GFG
{

    // Function to count
    // derangements
    static int countDer(int n)
    {
        // Create an array to store
        // counts for subproblems
        int []der = new int[n + 1];

        // Base cases
        der[0] = 1;
        der[1] = 0;
        der[2] = 1;

        // Fill der[0..n] in bottom up
        // manner using above recursive
        // formula
        for (int i = 3; i <= n; ++i)
            der[i] = (i - 1) * (der[i - 1] +
                                der[i - 2]);
        // Return result for n
        return der[n];
    }

    // Driver code
    public static void Main ()
    {
        int n = 4;
        Console.Write("Count of Derangements is " +
                       countDer(n));

    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// A Dynamic programming based PHP
// program to count derangements

function countDer($n)
{
    // Create an array to store
    // counts for subproblems

    // Base cases
    $der[0] = 1;
    $der[1] = 0;
    $der[2] = 1;

    // Fill der[0..n] in bottom up manner
    // using above recursive formula
    for ($i = 3; $i <= $n; ++$i)
        $der[$i] = ($i - 1) * ($der[$i - 1] +
                               $der[$i - 2]);

    // Return result for n
    return $der[$n];
}

// Driver code
$n = 4;
echo "Count of Derangements is ",
                    countDer($n);

// This code is contributed by aj_36
```

```
?>
```

Output :

```
Count of Derangements is 9
```

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

Thanks to Utkarsh Trivedi for suggesting above solution.

References:
https://en.wikipedia.org/wiki/Derangement

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/count-derangements-permutation-such-that-no-element-appears-in-its-original-posi

Chapter 15

Count Pairs Of Consecutive Zeros

Count Pairs Of Consecutive Zeros - GeeksforGeeks

Consider a sequence that starts with a 1 on a machine. At each successive step, the machine simultaneously transforms each digit 0 into the sequence 10 and each digit 1 into the sequence 01.
After the first time step, the sequence 01 is obtained; after the second, the sequence 1001, after the third, the sequence 01101001 and so on.

How many pairs of consecutive zeros will appear in the sequence after n steps?

Examples :

```
Input : Number of steps = 3
Output: 1
// After 3rd step sequence will be  01101001

Input : Number of steps = 4
Output: 3
// After 4rd step sequence will be 1001011001101001

Input : Number of steps = 5
Output: 5
// After 3rd step sequence will be  01101001100101101001011001101001
```

This is a simple reasoning problem. If we see the sequence very carefully , then we will be able to find a pattern for given sequence. If n=1 sequence will be {01} so number of pairs of consecutive zeros are 0, If n = 2 sequence will be {1001} so number of pairs of consecutive zeros are 1, If n=3 sequence will be {01101001} so number of pairs of consecutive zeros are 1,
If n=4 sequence will be {1001011001101001} so number of pairs of consecutive zeros are 3.

So length of the sequence will always be a power of 2. We can see after length 12 sequence is repeating and in lengths of 12. And in a segment of length 12, there are total 2 pairs of consecutive zeros. Hence we can generalize the given pattern q = (2^n/12) and total pairs of consecutive zeros will be 2*q+1.

C++

```
 // C++ program to find number of consecutive
// 0s in a sequence
#include<bits/stdc++.h>
using namespace std;

// Function to find number of consecutive Zero Pairs
// Here n is number of steps
int consecutiveZeroPairs(int n)
{
    // Base cases
    if (n==1)
        return 0;
    if (n==2 || n==3)
        return 1;

    // Calculating how many times divisible by 12, i.e.,
    // count total number repeating segments of length 12
    int q = (pow(2, n) / 12);

    // number of consecutive Zero Pairs
    return 2 * q + 1;
}

// Driver code
int main()
{
    int n = 5;
    cout << consecutiveZeroPairs(n) << endl;
    return 0;
}
```

Java

```
 //Java program to find number of
// consecutive 0s in a sequence
import java.io.*;
import java.math.*;

class GFG {
    // Function to find number of consecutive
    // Zero Pairs. Here n is number of steps
    static int consecutiveZeroPairs(int n)
    {
        // Base cases
        if (n == 1)
            return 0;
        if (n == 2 || n == 3)
            return 1;

        // Calculating how many times divisible
        // by 12, i.e.,count total number
        // repeating segments of length 12
        int q = ((int)(Math.pow(2, n)) / 12);

        // number of consecutive Zero Pairs
        return (2 * q + 1);
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 5;
        System.out.println(consecutiveZeroPairs(n));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python

```
 # Python program to find number of
# consecutive 0s in a sequence
import math

# Function to find number of consecutive
# Zero Pairs. Here n is number of steps
def consecutiveZeroPairs(n) :

    # Base cases
    if (n == 1) :
        return 0
    if (n == 2 or n == 3) :
        return 1

    # Calculating how many times divisible
    # by 12, i.e.,count total number
    # repeating segments of length 12
    q =(int) (pow(2,n) / 12)

    # number of consecutive Zero Pairs
    return 2 * q + 1

# Driver code
n = 5
print consecutiveZeroPairs(n)

#This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find number of
// consecutive 0s in a sequence
using System;

class GFG {

    // Function to find number of
    // consecutive Zero Pairs.
    // Here n is number of steps
    static int consecutiveZeroPairs(int n)
    {
        // Base cases
        if (n == 1)
            return 0;
        if (n == 2 || n == 3)
            return 1;

        // Calculating how many times divisible
        // by 12, i.e.,count total number
        // repeating segments of length 12
        int q = ((int)(Math.Pow(2, n)) / 12);

        // number of consecutive Zero Pairs
        return (2 * q + 1);
    }

    // Driver Code
    public static void Main()
    {
        int n = 5;
        Console.Write(consecutiveZeroPairs(n));
    }
}

// This code is contributed by Nitin mittal.
```

PHP

```
 <?php
// PHP program to find number
// of consecutive 0s in a sequence

// Function to find number
// of consecutive Zero Pairs
// Here n is number of steps
function consecutiveZeroPairs($n)
{
    // Base cases
    if ($n == 1)
        return 0;
    if ($n == 2 || $n == 3)
        return 1;

    // Calculating how many times
    // divisible by 12, i.e., count
    // total number repeating segments
    // of length 12
    $q = floor(pow(2, $n) / 12);

    // number of consecutive Zero Pairs
    return 2 * $q + 1;
}

// Driver code
$n = 5;
echo consecutiveZeroPairs($n) ;

// This code is contributed
// by nitin mittal.
?>
```

Output :

```
5
```

This article is contributed by **Shashank Mishra (Gullu)**. this article is reviewed by team GeeksForGeeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-pairs-of-consecutive-zeros/

Chapter 16

Count binary strings with twice zeros in first half

Count binary strings with twice zeros in first half - GeeksforGeeks

We are given an integer N. We need to count the total number of such binary strings of length N such that the number of '0's in the first string of length N/2 is double of the number of '0's in the second string of length N/2.
Note: N is always an even positive integer.

Examples:

```
Input : N = 4
Output : 2
Explanation: 0010 and 0001 are two binary string such that the number of zero
in the first half is double the number of zero in second half.

Input : N = 6
Output : 9
```

Naive Approach: We can generate all the binary string of length N and then one by one can check that whether selected string follows the given scenario. But the time complexity for this approach is exponential and is O(N*2^N).

Efficient Approach: This approach is based on some mathematical analysis of the problem.
Pre-requisite for this approach: Factorial and Combinatoric with modulo function.
Note: Let n = N/2.
If we perform some analysis step by step:

1. The number of strings which contain two '0' in first half string and one '0' in second half string, is equal to $\mathbf{{}^nC_2 * {}^nC_1}$.
2. The number of strings which contain four '0' in first half string and two '0' in second half string, is equal to $\mathbf{{}^nC_4 * {}^nC_2}$.

3. The number of strings which contain six '0' in first half string and three '0' in second half string, is equal to $\mathbf{{}^{n}C_{6} * {}^{n}C_{3}}$.

We repeat above procedure till the number of '0' in first half become n if n is even or (n-1) if n is odd.
So, our final answer is $\mathbf{\Sigma\ ({}^{n}C_{(2*i)} * {}^{n}C_{i})}$, for 2 <= 2*i <= n.

Now, we can compute the required number of strings just by using Permutation and Combination.

Algorithm :

```
int n = N/2;
for(int i=2;i<=n;i=i+2)
             ans += ( nCr(n, i) * nCr(n, i/2);
```

Note : You can use Dynamic Programming technique to pre-compute CoeFunc(N, i) i.e. $^{n}C_{i}$.

Time complexity is O(N) if we pre-compute CoeFunc(N, i) in O(N*N).

CPP

```
 // CPP for finding number of binary strings
// number of '0' in first half is double the
// number of '0' in second half of string
#include <bits/stdc++.h>

// pre define some constant
#define mod 1000000007
#define max 1001
using namespace std;

// global values for pre computation
long long int nCr[1003][1003];

void preComputeCoeff()
{
    for (int i = 0; i < max; i++) {
        for (int j = 0; j <= i; j++) {
            if (j == 0 || j == i)
                nCr[i][j] = 1;
            else
                nCr[i][j] = (nCr[i - 1][j - 1] +
                             nCr[i - 1][j]) % mod;
        }
    }
}

// function to print number of required string
long long int computeStringCount(int N)
{
    int n = N / 2;
    long long int ans = 0;

    // calculate answer using proposed algorithm
    for (int i = 2; i <= n; i += 2)
        ans = (ans + ((nCr[n][i] * nCr[n][i / 2])
                      % mod)) % mod;
    return ans;
}

// Driver code
int main()
{
    preComputeCoeff();
    int N = 3;
    cout << computeStringCount(N) << endl;
    return 0;
}
```

Java

```
 // Java program for finding number of binary strings
// number of '0' in first half is double the
// number of '0' in second half of string
class GFG {

    // pre define some constant
    static final long mod = 1000000007;
    static final long max = 1001;

    // global values for pre computation
    static long nCr[][] = new long[1003][1003];

    static void preComputeCoeff()
    {
        for (int i = 0; i < max; i++) {
            for (int j = 0; j <= i; j++) {
                if (j == 0 || j == i)
                    nCr[i][j] = 1;
                else
                    nCr[i][j] = (nCr[i - 1][j - 1] +
                                 nCr[i - 1][j]) % mod;
            }
        }
    }

    // function to print number of required string
    static long computeStringCount(int N)
    {
        int n = N / 2;
        long ans = 0;

        // calculate answer using proposed algorithm
        for (int i = 2; i <= n; i += 2)
            ans = (ans + ((nCr[n][i] * nCr[n][i / 2])
                        % mod)) % mod;
        return ans;
    }

    // main function
    public static void main(String[] args)
    {
        preComputeCoeff();
        int N = 3;
        System.out.println( computeStringCount(N) );

    }
}

// This code is contributed by Arnab Kundu.
```

C#

```
 // C# program for finding number of binary
// strings number of '0' in first half is
// double the number of '0' in second half
// of string
using System;

class GFG {

    // pre define some constant
    static long mod = 1000000007;
    static long max = 1001;

    // global values for pre computation
    static long [,]nCr = new long[1003,1003];

    static void preComputeCoeff()
    {
        for (int i = 0; i < max; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                if (j == 0 || j == i)
                    nCr[i,j] = 1;
                else
                    nCr[i,j] = (nCr[i - 1,j - 1]
                          + nCr[i - 1,j]) % mod;
            }
        }
    }

    // function to print number of required
    // string
    static long computeStringCount(int N)
    {
        int n = N / 2;
        long ans = 0;

        // calculate answer using proposed
        // algorithm
        for (int i = 2; i <= n; i += 2)
            ans = (ans + ((nCr[n,i]
                * nCr[n,i / 2]) % mod)) % mod;

        return ans;
    }

    // main function
    public static void Main()
    {
        preComputeCoeff();
        int N = 3;
        Console.Write( computeStringCount(N) );

    }
}

// This code is contributed by nitin mittal.
```

Output:

```
0
```

Improved By : andrew1234, nitin mittal

Source

https://www.geeksforgeeks.org/count-binary-strings-twice-zeros-first-half/

Chapter 17

Count maximum-length palindromes in a String

Count maximum-length palindromes in a String - GeeksforGeeks

Given a string, count how many maximum-length palindromes are present.(It need not be a substring)
Examples:

```
Input : str = "ababa"
Output: 2
Explanation :
palindromes of maximum of lenghts are  :
 "ababa", "baaab"

Input : str = "ababab"
Output: 4
Explanation :
palindromes of maximum of lenghts are  :
 "ababa", "baaab", "abbba", "babab"
```

Approach A palindrome can be reprsented as **"str + t + reverse(str)"**.
Note: **"t"** is empty for even length palindromic strings.

Calculate in how many ways "str" can be made and then multiply with "t" (number of single characters left out).

Let c_i be the number of occurrences of character in the string. Consider the following cases:
1. If c_i is even. Then a half of every maximum palindrome will contain exactly letters $f_i = c_i / 2$.
2.If c_i is odd. Then a half of every maximum palindrome will contain exactly letters $f_i = c_i – 1/ 2$.

Let k be the number of odd c_i. If k=0, the maximum palindromes length will be even; otherwise it will be odd and there will be exactly k possible middle letters i.e., we can set this letter to the middle of palindrome.

The number of permutations of n objects with n_1 identical objects of type 1, n_2 identical objects of type 2,, and n3 identical objects of type 3 is **n! / (n1! * n2! * n3!)** .
So here we have total number of characters as $f_a+f_b+f_a+$…….$+f_y+f_z$. So number of permutation is **$(f_a+f_b+f_a+$…….$+f_y+f_z+)!$ / $f_a!$ $f_b!$…$f_y!f_z!$.**

Now If K is not 0, it's obvious that the answer is k * $(f_a+f_b+f_a+$…….$+f_y+f_z+)!$ / $f_a!$ $f_b!$…$f_y!f_z!$

Below is the implementation of the above.

```
 // C++ implementation for counting
// maximum length palindromes
#include <bits/stdc++.h>
using namespace std;

// factorial of a number
int fact(int n)
{
    int ans = 1;
    for (int i = 1; i <= n; i++)
        ans = ans * i;
    return (ans);
}

// function to count maximum length palindromes.
int numberOfPossiblePallindrome(string str, int n)
{
    // Count number of occurrence
    // of a charterer in the string
    unordered_map<char, int> mp;
    for (int i = 0; i < n; i++)
        mp[str[i]]++;

    int k = 0;  // Count of singles
    int num = 0;  // numerator of result
    int den = 1;  // denominator of result
    int fi;
    for (auto it = mp.begin(); it != mp.end(); ++it)
    {
        // if frequency is even
        // fi = ci / 2
        if (it->second % 2 == 0)
            fi = it->second / 2;

        // if frequency is odd
        // fi = ci - 1 / 2.
        else
        {
            fi = (it->second - 1) / 2;
            k++;
        }

        // sum of all frequencies
        num = num + fi;

        // product of factorial of
        // every frequency
        den = den * fact(fi);
    }

    // if all character are unique
    // so there will be no pallindrome,
    // so if num != 0 then only we are
    // finding the factorial
    if (num != 0)
        num = fact(num);

    int ans = num / den;

    if (k != 0)
    {
        // k are the single
        // elements that can be
        // placed in middle
        ans = ans * k;
    }

    return (ans);
}

// Driver program
int main()
{
    char str[] = "ababab";
    int n = strlen(str);
    cout << numberOfPossiblePallindrome(str, n);
    return 0;
}
```

Output:

```
4
```

Time Complexity : O(n)

Source

https://www.geeksforgeeks.org/count-maximum-length-palindromes-string/

Chapter 18

Count number of equal pairs in a string

Count number of equal pairs in a string - GeeksforGeeks

Given a string s, find the number of pairs of characters that are same. Pairs (s[i], s[j]), (s[j], s[i]), (s[i], s[i]), (s[j], s[j]) should be considered different.

Examples :

```
Input: air
Output: 3
Explanation :
3 pairs that are equal are (a, a), (i, i) and (r, r)

Input : geeksforgeeks
Output : 31
```

The **naive approach** will be you to run two nested for loops and find out all pairs and keep a count of all pairs. But this is not efficient enough for longer length of strings.

For an **efficient approach**, we need to count the number of equal pairs in **linear time**. Since pairs (x, y) and pairs (y, x) are considered different. We need to use a hash table to store the count of all occurences of a character.So we know if a character occurs twice, then it will have 4 pairs – **(i, i), (j, j), (i, j), (j, i)**. So using a hash function, store the occurrence of each character, then for each character the number of pairs will be occurrence^2. Hash table will be 256 in length as we have 256 characters.

Below is the implementation of the above approach :

C++

```
 // CPP program to count the number of pairs
#include <bits/stdc++.h>
using namespace std;
#define MAX 256

// Function to count the number of equal pairs
int countPairs(string s)
{
    // Hash table
    int cnt[MAX] = { 0 };

    // Traverse the string and count occurrence
    for (int i = 0; i < s.length(); i++)
        cnt[s[i]]++;

    // Stores the answer
    int ans = 0;

    // Traverse and check the occurrence of every character
    for (int i = 0; i < MAX; i++)
        ans += cnt[i] * cnt[i];

    return ans;
}

// Driver Code
int main()
{
    string s = "geeksforgeeks";
    cout << countPairs(s);
    return 0;
}
```

Java

```
 // Java program to count the number of pairs
import java.io.*;

class GFG {

    static int MAX = 256;

    // Function to count the number of equal pairs
    static int countPairs(String s)
    {
        // Hash table
        int cnt[] = new int[MAX];
        // Traverse the string and count occurrence
        for (int i = 0; i < s.length(); i++)
            cnt[s.charAt(i)]++;

        // Stores the answer
        int ans = 0;

        // Traverse and check the occurrence
        // of every character
        for (int i = 0; i < MAX; i++)
            ans += cnt[i] * cnt[i];

        return ans;
    }

    // Driver Code
    public static void main (String[] args)
    {
        String s = "geeksforgeeks";
        System.out.println(countPairs(s));
    }
}

// This code is contributed by vt_m
```

C#

```
 // C# program to count the number of pairs
using System;

class GFG {

    static int MAX = 256;

    // Function to count the number of equal pairs
    static int countPairs(string s)
    {
        // Hash table
        int []cnt = new int[MAX];

        // Traverse the string and count occurrence
        for (int i = 0; i < s.Length; i++)
            cnt[s[i]]++;

        // Stores the answer
        int ans = 0;

        // Traverse and check the occurrence
        // of every character
        for (int i = 0; i < MAX; i++)
            ans += cnt[i] * cnt[i];

        return ans;
    }

    // Driver Code
    public static void Main ()
    {
        string s = "geeksforgeeks";
        Console.WriteLine(countPairs(s));
    }
}

// This code is contributed by vt_m
```

Output :

```
31
```

Source

https://www.geeksforgeeks.org/number-equal-pairs-string/

Chapter 19

Count number of strings (made of R, G and B) using given combination

Count number of strings (made of R, G and B) using given combination - GeeksforGeeks

We need to make a string of size n. Each character of the string is either 'R', 'B' or 'G'. In the final string there needs to be at least r number of 'R', at least b number of 'B' and at least g number of 'G' (such that r + g + b <= n). We need to find number of such strings possible.

Examples:

```
Input : n = 4, r = 1,
        b = 1, g = 1.
Output: 36
No. of 'R' >= 1,
No. of ‘G’ >= 1,
No. of ‘B’ >= 1 and
(No. of ‘R’) + (No. of ‘B’) + (No. of ‘G’) = n
then following cases are possible:
1. RBGR and its 12 permutation
2. RBGB and its 12 permutation
3. RBGG and its 12 permutation
Hence answer is 36.
```

Asked in : Directi

1. As R, B and G have to be included atleast for given no. of times. Remaining values = n -(r + b + g).

2. Make all combinations for the remaining values.
3. Consider each element one by one for the remaining values and sum up all the permutations.
4. Return total no. of permutations of all the combinations.

C++

```
 // C++ program to count number of possible strings
// with n characters.
#include<bits/stdc++.h>
using namespace std;

// Function to calculate number of strings
int possibleStrings( int n, int r, int b, int g)
{
    // Store factorial of numbers up to n
    // for further computation
    int fact[n+1];
    fact[0] = 1;
    for (int i = 1; i <= n; i++)
        fact[i] = fact[i-1] * i;

    // Find the remaining values to be added
    int left = n - (r+g+b);
    int sum = 0;

    // Make all possible combinations
    // of R, B and G for the remaining value
    for (int i = 0; i <= left; i++)
    {
        for (int j = 0; j<= left-i; j++)
        {
            int k = left - (i+j);

            // Compute permutation of each combination
            // one by one and add them.
            sum = sum + fact[n] /
                           (fact[i+r]*fact[j+b]*fact[k+g]);
        }
    }

    // Return total no. of strings/permutation
    return sum;
}

// Drivers code
int main()
{
    int n = 4, r = 2;
    int b = 0, g = 1;
    cout << possibleStrings(n, r, b, g);
    return 0;
}
```

PHP

```
 <?php
// PHP program to count number
// of possible strings with
// n characters.

// Function to calculate
// number of strings
function possibleStrings( $n, $r, $b, $g)
{

    // Store factorial of
    // numbers up to n for
    // further computation
    $fact[0] = 1;
    for ($i = 1; $i <= $n; $i++)
        $fact[$i] = $fact[$i - 1] * $i;

    // Find the remaining
    // values to be added
    $left = $n - ($r + $g + $b);
    $sum = 0;

    // Make all possible combinations
    // of R, B and G for the remaining value
    for ($i = 0; $i <= $left; $i++)
    {
        for ($j = 0; $j <= $left - $i; $j++)
        {
            $k = $left - ($i+$j);

            // Compute permutation of
            // each combination one
            // by one and add them.
            $sum = $sum + $fact[$n] /
                   ($fact[$i + $r] *
                   $fact[$j + $b] *
                   $fact[$k + $g]);
        }
    }
    // Return total no. of
    // strings/permutation
    return $sum;
}

    // Driver Code
    $n = 4; $r = 2;
    $b = 0; $g = 1;

    echo possibleStrings($n, $r, $b, $g);

// This code is contributed by jit_t.
?>
```

Output:

```
22
```

To handle n with large numbers, we can use the concept of Large Factorial.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-number-of-strings-made-of-r-g-and-b-using-given-combination/

Chapter 20

Count number of triplets with product equal to given number with duplicates allowed

Count number of triplets with product equal to given number with duplicates allowed - GeeksforGeeks

Given an array of positive integers (may contain **duplicates**), the task is to find the number of triplets whose product is equal to a given number **t**.

Examples:

```
Input: arr = [1, 31, 3, 1, 93, 3, 31, 1, 93]
        t = 93
Output: 18

Input: arr = [4, 2, 4, 2, 3, 1]
        t = 8
Output: 4
[(4, 2, 1), (4, 2, 1), (2, 4, 1), (4, 2, 1)]
```

Naive Approach: The easiest way to solve this is to compare each possible triplet with **t** and increment count if their product is equal to **t**.

Below is the implementation of above approach:

C++

```
 // C++ program for above implementation
#include<iostream>
```

```
using namespace std ;

int main()
{
    // The target value for which
    // we have to find the solution
    int target = 93 ;

    int arr[] = {1, 31, 3, 1, 93,
                    3, 31, 1, 93};
    int length = sizeof(arr) /
                 sizeof(arr[0]) ;

    // This variable contains the total
    // count of triplets found
    int totalCount = 0 ;

    // Loop from the first to the third
    //last integer in the list
    for(int i = 0 ; i < length - 2; i++)
    {
        // Check if arr[i] is a factor
        // of target or not. If not,
        // skip to the next element
        if (target % arr[i] == 0)
        {
            for (int j = i + 1 ;
                     j < length - 1; j++)
            {
            // Check if the pair (arr[i], arr[j])
            // can be a part of triplet whose
            // product is equal to the target
            if (target % (arr[i] * arr[j]) == 0)
                {
                // Find the remaining
                // element of the triplet
                int toFind = target / (arr[i] * arr[j]) ;

                    for(int k = j + 1 ; k < length ; k++ )
                    {
                        // If element is found. increment
                        // the total count of the triplets
                        if (arr[k] == toFind)
                        {
                            totalCount ++ ;
                        }
                    }
                }
            }
        }
    }
cout << "Total number of triplets found : "
     << totalCount ;

return 0 ;
}

// This code is contributed by ANKITRAI1
```

Java

// Java program for above implementation
class GFG
{
public static void main(String[] args)
{
// The target value for which
// we have to find the solution
int target = 93 ;

int[] arr = {1, 31, 3, 1, 93,
3, 31, 1, 93};
int length = arr.length;

// This variable contains the total
// count of triplets found
int totalCount = 0 ;

// Loop from the first to the third
//last integer in the list
for(int i = 0 ; i < length - 2; i++) { // Check if arr[i] is a factor // of target or not. If not, // skip to the next element if (target % arr[i] == 0) { for (int j = i + 1 ; j < length - 1; j++) { // Check if the pair (arr[i], arr[j]) // can be a part of triplet whose // product is equal to the target if (target % (arr[i] * arr[j]) == 0) { // Find the remaining // element of the triplet int toFind = target / (arr[i] * arr[j]); for(int k = j + 1 ; k < length ; k++) { // If element is found. increment // the total count of the triplets if (arr[k] == toFind) { totalCount ++ ; } } } } } } System.out.println("Total number of triplets found : " + totalCount); } } // This code is contributed by mits [tabby title="Python3"]

```
 # Python program for above implementation

# The target value for which we have
# to find the solution
target = 93

arr = [1, 31, 3, 1, 93, 3, 31, 1, 93]
length = len(arr)
```

```
# This variable contains the total
# count of triplets found
totalCount = 0

# Loop from the first to the third
# last integer in the list
for i in range(length - 2):

    # Check if arr[i] is a factor of target
    # or not. If not, skip to the next element
    if target % arr[i] == 0:
        for j in range(i + 1, length - 1):

            # Check if the pair (arr[i], arr[j]) can be
            # a part of triplet whose product is equal
            # to the target
            if target % (arr[i] * arr[j]) == 0:

                # Find the remaining element of the triplet
                toFind = target // (arr[i] * arr[j])
                for k in range(j + 1, length):

                    # If element is found. increment the
                    # total count of the triplets
                    if arr[k] == toFind:
                        totalCount += 1

print ('Total number of triplets found: ', totalCount)
```

Output:

```
Total number of triplets found:  18
```

Time Complexity: O(n^3)

Efficient Approach:

1. Remove the numbers which are not the factors of **t** from the array.
2. Then sort the array so that we don't have to verify the index of each number to avoid additional counting of pairs.
3. Then store the number of times each number appears in a dictionary **count**.
4. Use two loops to find the first two numbers of a valid triplet by checking if their product divides **t**

5. Find the third number of the triplet and check if we have already seen the triplet to avoid duplicate calculations
6. Count the total possible combinations of that triplet such that they occur in the same order (all pairs should follow the order (x, y, z) to avoid repititions)

Python3

```
 # Python3 code to find the number of triplets
# whose product is equal to a given number
# in quadratic time

# This function is used to initialize
# a dictionary with a default value
from collections import defaultdict

# Value for which solution has to be found
target = 93
arr = [1, 31, 3, 1, 93, 3, 31, 1, 93]

# Create a list of integers from arr which
# contains only factors of the target
# Using list comprehension
arr = [x for x in arr if x != 0 and target % x == 0]

# Sort the list
arr.sort()
length = len(arr)

# Initialize the dictionary with the default value
tripletSeen = defaultdict(lambda : False)
count = defaultdict(lambda : 0)

# Count the number of times a value is present in
# the list and store it in a dictionary for further use
for key in arr:
    count[key] += 1

# Used to store the total number of triplets
totalCount = 0

# This function returns the total number of combinations
# of the triplet (x, y, z) possible in the given list
def Combinations(x, y, z):

    if x == y:
        if y == z:
            return (count[x]*(count[y]-1)*(count[z]-2)) // 6
        else:
            return ((((count[y]-1)*count[x]) // 2)*count[z])

    elif y == z:
        return count[x]*(((count[z]-1)*count[y]) // 2)

    else:
        return (count[x] * count[y] * count[z])

for i in range(length - 2):
    for j in range(i + 1, length - 1):

        # Check if the pair (arr[i], arr[j]) can be a
        # part of triplet whose product is equal to the target
        if target % (arr[i] * arr[j]) == 0:
            toFind = target // (arr[i] * arr[j])

            # This condition makes sure that a solution is not repeated
            if (toFind >= arr[i] and toFind >= arr[j] and
                tripletSeen[(arr[i], arr[j], toFind)] == False):

                tripletSeen[(arr[i], arr[j], toFind)] = True
                totalCount += Combinations(arr[i], arr[j], toFind)

print ('Total number of triplets found: ', totalCount)
```

Output:

```
Total number of triplets found:  18
```

Time Complexity: O(n^2)

Improved By : ANKITRAI1, Mithun Kumar

Source

https://www.geeksforgeeks.org/count-number-of-triplets-with-product-equal-to-given-number-with-duplicates-allo

Chapter 21

Count number of triplets with product equal to given number with duplicates allowed | Set-2

Count number of triplets with product equal to given number with duplicates allowed | Set-2 - GeeksforGeeks

Given an array of positive integers(may contain duplicates) and a number 'm', find the number of unordered triplets **((A_i, A_j, A_k) and (A_j, A_i, A_k) and other permutations are counted as one only)** with product equal to 'm'.

Examples:

> **Input:** arr[] = { 1, 4, 6, 2, 3, 8}, M = 24
> **Output:** 3
> The triplets are {1, 4, 6} {1, 3, 8} {4, 2, 3}
>
> **Input:** arr[] = { 0, 4, 6, 2, 3, 8}, M = 18
> **Output:** 0
> There are no triplets in this case

A solution with O(N^2) has been discussed in the previous post. In this post a better approach with lesser complexity has been discussed.

Approach: The below algorithm is followed to solve the above problem.

- Use a hash-map to count the frequency of every element in the given array.
- Declare a set which can store triplets, so that only unordered triplets are taken to count.
- Iterate from 1 to sqrt(m) in a loop(let variable be i), since the maximum number by which M is divisible is sqrt(M) leaving out M.

- Check if M is divisible by i or not and i is present in the array of integers or not, if it is, then again loop from 1 to M/i.(let the loop variable be j).
- Again Check if M is divisible by j or not and j is present in the array of integers or not, if it is then check if the remaining number that is **((M / i) / j)** is present or not.
- If it is present, then a triplet has been formed. To avoid duplicate triplets, insert them in the set in a sorted order.
- Check if the set the size increases after the insertion of triplet, if it does then use combinatorics to find the number of triplets.
- To find the number of triplets, the following conditions will be there.

  1. If all of the A_i, A_j and A_k are unique, then number of combinations will be the prthe oduct of their frequencies.
  2. If all of them are same, then we can only choose three of them, hence the formula stands at [illegible].
  3. If any of the two are same(let Ai and Aj), the count will be [illegible] * frequency[A_k]

Below is the implementation of the above approach.

```
 // C++ program to find the
// number of triplets in array
// whose product is equal to M
#include <bits/stdc++.h>
using namespace std;

// Function to count the triplets
int countTriplets(int a[], int m, int n)
{

    // hash-map to store the frequency of every number
    unordered_map<int, int> frequency;

    // set to store the unique triplets
    set<pair<int, pair<int, int> > > st;

    // count the number of times
    // every elememt appears in a map
    for (int i = 0; i < n; i++) {
        frequency[a[i]] += 1;
    }

    // stores the answer
    int ans = 0;
    // iterate till sqrt(m) since tnum2t is the
    // mamimum number tnum2t can divide M except itself
    for (int i = 1; i * i <= m; i++) {

        // if divisible and present
        if (m % i == 0 and frequency[i]) {

            // remaining number after divison
            int num1 = m / i;

            // iterate for the second number of the triplet
            for (int j = 1; j * j <= num1; j++) {

                // if divisible and present
                if (num1 % j == 0 and frequency[j]) {

                    // remaining number after divison
                    int num2 = num1 / j;

                    // if the third number is present in array
                    if (frequency[num2]) {

                        // a temp array to store the triplet
                        int temp[] = { num2, i, j };

                        // sort the triplets
                        sort(temp, temp + 3);

                        // get the size of set
                        int setsize = st.size();

                        // insert the triplet in ascending order
                        st.insert({ temp[0], { temp[1], temp[2] } });

                        // if the set size does not increase after
                        // insertion, it means a new triplet is found
                        if (setsize != st.size()) {

                            // if all the number in triplets are unique
                            if (i != j and j != num2)
                                ans += frequency[i] * frequency[j] * frequency[num2];

                            // if Ai and Aj are same among triplets
                            else if (i == j && j != num2)
                                ans += (frequency[i] * (frequency[i] - 1) / 2)
                                       * frequency[num2];

                            // if Aj and Ak are same among triplets
                            else if (j == num2 && j != i)
                                ans += (frequency[j] * (frequency[j] - 1) / 2)
                                       * frequency[i];

                            // if three of them are
                            // same among triplets
                            else if (i == j and j == num2)
                                ans += (frequency[i] * (frequency[i] - 1) * (frequency[i] - 2) /

                            // if Ai and Ak are same among triplets
                            else
                                ans += (frequency[i] * (frequency[i] - 1) / 2)
                                       * frequency[j];
                        }
                    }
                }
            }
        }
    }

    return ans;
}

// Driver Code
int main()
{
    int a[] = { 1, 4, 6, 2, 3, 8 };
    int m = 24;
    int n = sizeof(a) / sizeof(a[0]);

    cout << countTriplets(a, m, n);

    return 0;
}
```

Output:

```
3
```

Time Complexity: O(N * log N)

Source

https://www.geeksforgeeks.org/count-number-of-triplets-with-product-equal-to-given-number-with-duplicates-allo

Chapter 22

Count of arrays having consecutive element with different values

Count of arrays having consecutive element with different values - GeeksforGeeks

Given three positive integers **n**, **k** and **x**. The task is to count the number of different array that can be formed of size n such that each element is between 1 to k and two consecutive element are different. Also, the first and last elements of each array should be 1 and x respectively.

Examples :

```
Input : n = 4, k = 3, x = 2
Output : 3
```

The idea is to use Dynamic Programming and combinatorics to solve the problem.
First of all, notice that the answer is same for all x from 2 to k. It can easily be proved. This will be useful later on.
Let the state f(i) denote the number of ways to fill the range [1, i] of array A such that $A_1 = 1$ and A_i 1.
Therefore, if x   1, the answer to the problem is f(n)/(k – 1), because f(n) is the number of way where A_n is filled with a number from 2 to k, and the answer are equal for all such values A_n, so the answer for an individual value is f(n)/(k – 1).
Otherwise, if x = 1, the answer is f(n – 1), because A_{n-1} 1, and the only number we can fill A_n with is x = 1.
Now, the main problem is how to calculate f(i). Consider all numbers that A_{i-1} can be. We know that it must lie in [1, k].

- If A_{i-1} 1, then there are (k – 2)f(i – 1) ways to fill in the rest of the array, because A_i cannot be 1 or A_{i-1} (so we multiply with (k – 2)), and for the range [1, i – 1], there are, recursively, f(i – 1) ways.
- If $A_{i-1} = 1$, then there are (k – 1)f(i – 2) ways to fill in the rest of the array, because $A_{i-1} = 1$ means A_{i-2} 1 which means there are f(i – 2)ways to fill in the range [1, i – 2] and the only value that A_i cannot be 1, so we have (k – 1) choices for A_i.

By combining the above, we get

```
f(i) = (k - 1) * f(i - 2) + (k - 2) * f(i - 1)
```

This will help us to use dynamic programming using f(i).

Below is the implementation of this approach:

C++

```
 // CPP Program to find count of arrays.
#include <bits/stdc++.h>
#define MAXN 109
using namespace std;

// Return the number of arrays with given constartints.
int countarray(int n, int k, int x)
{
    int dp[MAXN] = { 0 };

    // Initalising dp[0] and dp[1].
    dp[0] = 0;
    dp[1] = 1;

    // Computing f(i) for each 2 <= i <= n.
    for (int i = 2; i < n; i++)
        dp[i] = (k - 2) * dp[i - 1] +
                (k - 1) * dp[i - 2];

    return (x == 1 ? (k - 1) * dp[n - 2] : dp[n - 1]);
}

// Driven Program
int main()
{
    int n = 4, k = 3, x = 2;
    cout << countarray(n, k, x) << endl;
    return 0;
}
```

Java

```
 // Java program to find count of arrays.
import java.util.*;

class Counting
{
    static int MAXN = 109;

    public static int countarray(int n, int k,
                                   int x)
    {
        int[] dp = new int[109];

        // Initalising dp[0] and dp[1].
        dp[0] = 0;
        dp[1] = 1;

        // Computing f(i) for each 2 <= i <= n.
        for (int i = 2; i < n; i++)
            dp[i] = (k - 2) * dp[i - 1] +
                (k - 1) * dp[i - 2];

        return (x == 1 ? (k - 1) * dp[n - 2] :
                              dp[n - 1]);
    }

    // driver code
    public static void main(String[] args)
    {
        int n = 4, k = 3, x = 2;
        System.out.println(countarray(n, k, x));
    }
}


// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 code to find count of arrays.

# Return the number of lists with
# given constraints.
def countarray( n , k , x ):

    dp = list()

    # Initalising dp[0] and dp[1]
    dp.append(0)
    dp.append(1)

    # Computing f(i) for each 2 <= i <= n.
    i = 2
    while i < n:
        dp.append( (k - 2) * dp[i - 1] +
                   (k - 1) * dp[i - 2])
        i = i + 1

    return ( (k - 1) * dp[n - 2] if x == 1 else dp[n - 1])

# Driven code
n = 4
k = 3
x = 2
print(countarray(n, k, x))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to find count of arrays.
using System;

class GFG
{
// static int MAXN = 109;

    public static int countarray(int n, int k,
                                int x)
    {
        int[] dp = new int[109];

        // Initalising dp[0] and dp[1].
        dp[0] = 0;
        dp[1] = 1;

        // Computing f(i) for each 2 <= i <= n.
        for (int i = 2; i < n; i++)
            dp[i] = (k - 2) * dp[i - 1] +
                    (k - 1) * dp[i - 2];

        return (x == 1 ? (k - 1) * dp[n - 2] :
                              dp[n - 1]);
    }

    // Driver code
    public static void Main()
    {
        int n = 4, k = 3, x = 2;
        Console.WriteLine(countarray(n, k, x));
    }
}


// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP Program to find
// count of arrays.

$MAXN = 109;

// Return the number of arrays
// with given constartints.
function countarray($n, $k, $x)
{
    $dp = array( 0 );

    // Initalising dp[0] and dp[1].
    $dp[0] = 0;
    $dp[1] = 1;

    // Computing f(i) for
    // each 2 <= i <= n.
    for ( $i = 2; $i < $n; $i++)
        $dp[$i] = ($k - 2) * $dp[$i - 1] +
                  ($k - 1) * $dp[$i - 2];

    return ($x == 1 ? ($k - 1) *
            $dp[$n - 2] : $dp[$n - 1]);
}

// Driven Code
$n = 4; $k = 3; $x = 2;
echo countarray($n, $k, $x) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
3
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-arrays-consecutive-element-different-values/

Chapter 23

Count of different straight lines with total n points with m collinear

Count of different straight lines with total n points with m collinear - GeeksforGeeks

There are 'n' points in a plane out of which 'm points are collinear. How many different straight lines can form?

Examples:

```
Input : n = 3, m = 3
Output : 1
We can form only 1 distinct straight line
using 3 collinear points

Input : n = 10, m = 4
Output : 40
```

> Number of distinct Straight lines = $\mathbf{{}^{n}C_{2} - {}^{m}C_{2} + 1}$
>
> **How does this formula work?**
> Consider the second example above. There are 10 points, out of which 4 collinear. A straight line will be formed by any two of these ten points. Thus forming a straight line amounts to selecting any two of the 10 points. Two points can be selected out of the 10 points in ${}^{n}C_{2}$ ways.
>
> Number of straight line formed by 10 points when no 2 of them are co-linear = $\mathbf{{}^{10}C_{2}}$……..(i)
> Similarly, the number of straight lines formed by 4 points when no 2 of them are co-linear = $\mathbf{{}^{4}C_{2}}$….(ii)

Since straight lines formed by these 4 points are sane, straight lines formed by them will reduce to only one.
Required number of straight lines formed = $^{10}C_2 - {}^{4}C_2 + 1 = 45 - 6 + 1 = 40$

Implementation of the approach is given as:

C++

```
 // CPP program to count number of straight lines
// with n total points, out of which m are
// collinear.
#include <bits/stdc++.h>

using namespace std;

// Returns value of binomial coefficient
// Code taken from https://goo.gl/vhy4jp
int nCk(int n, int k)
{

    int C[k+1];
    memset(C, 0, sizeof(C));

    C[0] = 1;  // nC0 is 1

    for (int i = 1; i <= n; i++)
    {

        // Compute next row of pascal triangle
        // using the previous row
        for (int j = min(i, k); j > 0; j--)

            C[j] = C[j] + C[j-1];
    }

    return C[k];
}


/* function to calculate number of straight lines
   can be formed */
int count_Straightlines(int n,int m)
{

    return (nCk(n, 2) - nCk(m, 2)+1);

}
```

```
/* driver function*/
int main()
{

    int n = 4, m = 3 ;
    cout << count_Straightlines(n, m);
    return 0;

}
```

Java

```
 // Java program to count number of straight lines
// with n total points, out of which m are
// collinear.
import java.util.*;
import java.lang.*;

public class GfG  {

    // Returns value of binomial coefficient
    // Code taken from https://goo.gl/vhy4jp
    public static int nCk(int n, int k)
    {
        int[] C = new int[k + 1];

        C[0] = 1; // nC0 is 1

        for (int i = 1; i <= n; i++)  {

            // Compute next row of pascal triangle
            // using the previous row
            for (int j = Math.min(i, k); j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }

        return C[k];
    }


    /* function to calculate number of straight lines
    can be formed */
    public static int count_Straightlines(int n, int m)
    {
        return (nCk(n, 2) - nCk(m, 2) + 1);
    }
    // Driver function
    public static void main(String argc[])
    {
        int n = 4, m = 3;
        System.out.println(count_Straightlines(n, m));
    }

    // This code is contributed by Sagar Shukla
}
```

Python

```
 # Python program to count number of straight lines
# with n total points, out of which m are
# collinear.

# Returns value of binomial coefficient
# Code taken from https://goo.gl/vhy4jp
def nCk(n, k):

    C = [0]* (k+1)

    C[0] = 1 # nC0 is 1

    for i in range(1, n+1):

        # Compute next row of pascal triangle
        # using the previous row
        j = min(i, k)

        while(j>0):

            C[j] = C[j] + C[j-1]
            j = j - 1

    return C[k]

#function to calculate number of straight lines
# can be formed
def count_Straightlines(n, m):

    return (nCk(n, 2) - nCk(m, 2)+1)

# Driven code
n = 4
m = 3
```

```
print( count_Straightlines(n, m) );

# This code is contributed by "rishabh_jain".
```

C#

```
 // C# program to count number of straight
// lines with n total points, out of
// which m are collinear.
using System;

public class GfG {

    // Returns value of binomial coefficient
    // Code taken from https://goo.gl/vhy4jp
    public static int nCk(int n, int k)
    {
        int[] C = new int[k + 1];

        // nC0 is 1
        C[0] = 1;

        for (int i = 1; i <= n; i++)
        {

            // Compute next row of pascal triangle
            // using the previous row
            for (int j = Math.Min(i, k); j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }

        return C[k];
    }


    // Function to calculate number of
    // straight lines can be formed
    public static int count_Straightlines(int n, int m)
    {
        return (nCk(n, 2) - nCk(m, 2) + 1);
    }


    // Driver Code
    public static void Main(String []args)
    {
        int n = 4, m = 3;
        Console.WriteLine(count_Straightlines(n, m));
    }


}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count number of straight lines
// with n total points, out of which m are
// collinear.

// Returns value of binomial coefficient
function nCk($n, $k)
{
    $C = array_fill(0, $k + 1, NULL);

    $C[0] = 1; // nC0 is 1

    for ($i = 1; $i <= $n; $i++)
    {

        // Compute next row of pascal triangle
        // using the previous row
        for ($j = min($i, $k); $j > 0; $j--)
            $C[$j] = $C[$j] + $C[$j-1];
    }
    return $C[$k];
}


// function to calculate
// number of straight lines
// can be formed
function count_Straightlines($n, $m)
{

    return (nCk($n, 2) - nCk($m, 2) + 1);

}

// Driver Code
$n = 4;
$m = 3;
echo(count_Straightlines($n, $m));
```

```
// This code is contributed
// by Prasad Kshirsagar
?>
```

Output:

```
4
```

Improved By : vt_m, Prasad_Kshirsagar, NishantBaranwalSomy

Source

https://www.geeksforgeeks.org/count-different-straight-lines-total-n-points-m-collinear/

Chapter 24

Count of different ways to express N as the sum of 1, 3 and 4

Count of different ways to express N as the sum of 1, 3 and 4 - GeeksforGeeks

Given N, count the number of ways to express N as sum of 1, 3 and 4.

Examples:

```
Input :  N = 4
Output : 4
Explanation: 1+1+1+1
             1+3
             3+1
             4

Input : N = 5
Output : 6
Explanation: 1 + 1 + 1 + 1 + 1
             1 + 4
             4 + 1
             1 + 1 + 3
             1 + 3 + 1
             3 + 1 + 1
```

Approach : Divide the problem into sub-problems for solving it. Let DP[n] be the be the number of ways to write N as the sum of 1, 3, and 4. Consider one possible solution with n = x1 + x2 + x3 + … xn. If the last number is 1, then sum of the remaining numbers is n-1. So the number that ends with 1 is equal to DP[n-1]. Taking other cases into account where the last number is 3 and 4. The final recurrence would be:

```
DPn = DPn-1 + DPn-3 + DPn-4
```

```
Base case :
DP[0] = DP[1] = DP[2] = 1 and DP[3] = 2
```

C++

```
 // CPP program to illustrate the number of
// ways to represent N as sum of 1, 3 and 4.
#include <bits/stdc++.h>
using namespace std;

// function to count the number of
// ways to represent n as sum of 1, 3 and 4
int countWays(int n)
{
    int DP[n + 1];

    // base cases
    DP[0] = DP[1] = DP[2] = 1;
    DP[3] = 2;

    // iterate for all values from 4 to n
    for (int i = 4; i <= n; i++)
        DP[i] = DP[i - 1] + DP[i - 3] + DP[i - 4];

    return DP[n];
}

// driver code
int main()
{
    int n = 10;
    cout << countWays(n);
    return 0;
}
```

Java

```
 // Java program to illustrate
// the number of ways to represent
// N as sum of 1, 3 and 4.

class GFG {
    // Function to count the
    // number of ways to represent
    // n as sum of 1, 3 and 4
    static int countWays(int n)
    {
        int DP[] = new int[n + 1];

        // base cases
        DP[0] = DP[1] = DP[2] = 1;
        DP[3] = 2;

        // iterate for all values from 4 to n
        for (int i = 4; i <= n; i++)
            DP[i] = DP[i - 1] + DP[i - 3]
                    + DP[i - 4];

        return DP[n];
    }

    // driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println(countWays(n));
    }
}

// This code is contributed
// by prerna saini.
```

Python3

```
 # Python program to illustrate the number of
# ways to represent N as sum of 1, 3 and 4.

# Function to count the number of
# ways to represent n as sum of 1, 3 and 4
def countWays(n):

    DP = [0 for i in range(0, n + 1)]

    # base cases
    DP[0] = DP[1] = DP[2] = 1
    DP[3] = 2

    # Iterate for all values from 4 to n
    for i in range(4, n + 1):
        DP[i] = DP[i - 1] + DP[i - 3] + DP[i - 4]
    return DP[n]


# Driver code
n = 10
print (countWays(n))

# This code is contributed by Gitanjali.
```

C#

```
 // C# program to illustrate
// the number of ways to represent
// N as sum of 1, 3 and 4.
using System;

class GFG {

    // Function to count the
    // number of ways to represent
    // n as sum of 1, 3 and 4
    static int countWays(int n)
    {
        int []DP = new int[n + 1];

        // base cases
        DP[0] = DP[1] = DP[2] = 1;
        DP[3] = 2;

        // iterate for all values from 4 to n
        for (int i = 4; i <= n; i++)
            DP[i] = DP[i - 1] + DP[i - 3]
                    + DP[i - 4];

        return DP[n];
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        Console.WriteLine(countWays(n));
    }
}

// This code is contributed
// by vt_m.
```

PHP

```
 <?php
// PHP program to illustrate
// the number of ways to
// represent N as sum of
// 1, 3 and 4.

// function to count the
// number of ways to
// represent n as sum of
// 1, 3 and 4
function countWays($n)
{
    $DP = array();

    // base cases
    $DP[0] = $DP[1] = $DP[2] = 1;
    $DP[3] = 2;

    // iterate for all
    // values from 4 to n
    for ($i = 4; $i <= $n; $i++)
        $DP[$i] = $DP[$i - 1] +
                  $DP[$i - 3] +
                  $DP[$i - 4];

    return $DP[$n];
}

// Driver Code
$n = 10;
echo countWays($n);

// This code is contributed
// by Sam007
?>
```

Output:

```
64
```

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

Improved By : Sam007

Source

https://www.geeksforgeeks.org/count-ofdifferent-ways-express-n-sum-1-3-4/

Chapter 25

Count of subsequences having maximum distinct elements

Count of subsequences having maximum distinct elements - GeeksforGeeks

Given an **arr** of size **n**. The problem is to count all the subsequences having maximum number of distinct elements.

Examples:

```
Input : arr[] = {4, 7, 6, 7}
Output : 2
The indexes for the subsequences are:
{0, 1, 2} - Subsequence is {4, 7, 6} and
{0, 2, 3} - Subsequence is {4, 6, 7}

Input : arr[] = {9, 6, 4, 4, 5, 9, 6, 1, 2}
Output : 8
```

Naive Approach: Consider all the subsequences having distinct elements and count the one's having maximum distinct elements.

Efficient Approach: Create a hash table to store the frequency of each element of the array. Take product of all the frequencies.
The solution is based on the fact that there is always 1 subsequence possible when all elements are distinct. If elements repeat, every occurrence of repeating element makes a mew subsequence of distinct elements.

```
 // C++ implementation to count subsequences having
// maximum distinct elements
#include <bits/stdc++.h>
using namespace std;
```

```
typedef unsigned long long int ull;

// function to count subsequences having
// maximum distinct elements
ull countSubseq(int arr[], int n)
{
    // unordered_map 'um' implemented as
    // hash table
    unordered_map<int, int> um;

    ull count = 1;

    // count frequency of each element
    for (int i = 0; i < n; i++)
        um[arr[i]]++;

    // traverse 'um'
    for (auto itr = um.begin(); itr != um.end(); itr++)

        // multiply frequency of each element
        // and accumulate it in 'count'
        count *= (itr->second);

    // required number of subsequences
    return count;
}

// Driver program to test above
int main()
{
    int arr[] = { 4, 7, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Count = "
         << countSubseq(arr, n);
    return 0;
}
```

Output:

```
Count = 2
```

Time Complexity: O(n).
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/count-subsequences-maximum-distinct-elements/

Chapter 26

Count of triangles with total n points with m collinear

Count of triangles with total n points with m collinear - GeeksforGeeks

There are 'n' points in a plane, out of which 'm' points are co-linear. Find the number of triangles formed by the points as vertices ?

Examples :

```
Input :  n = 5, m = 4
Output : 6
Out of five points, four points are
collinear, we can make 6 triangles. We
can choose any 2 points from 4 collinear
points and use the single point as 3rd
point. So total count is 4C2 = 6

Input :  n = 10, m = 4
Output : 116
```

Number of triangles = $\mathbf{{}^{n}C_{3} - {}^{m}C_{3}}$

How does this formula work?
Consider the second example above. There are 10 points, out of which 4 collinear. A triangle will be formed by any three of these ten points. Thus forming a triangle amounts to selecting any three of the 10 points. Three points can be selected out of the 10 points in ${}^{n}C_{3}$ ways.
Number of triangles formed by 10 points when no 3 of them are co-linear = ${}^{10}C_{3}$……(i)
Similarly, the number of triangles formed by 4 points when no 3 of them are co-linear = ${}^{4}C_{3}$……..(ii)

Since triangle formed by these 4 points are not valid, required number of triangles formed = ${}^{10}C_3$ – ${}^{4}C_3$ = 120 – 4 = 116

C++

```
 // CPP program to count number of triangles
// with n total points, out of which m are
// collinear.
#include <bits/stdc++.h>
using namespace std;

// Returns value of binomial coefficient
// Code taken from https://goo.gl/vhy4jp
int nCk(int n, int k)
{
    int C[k+1];
    memset(C, 0, sizeof(C));

    C[0] = 1;  // nC0 is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal triangle
        // using the previous row
        for (int j = min(i, k); j > 0; j--)
            C[j] = C[j] + C[j-1];
    }
    return C[k];
}

/* function to calculate number of triangle
   can be formed */
int counTriangles(int n,int m)
{
    return (nCk(n, 3) - nCk(m, 3));
}

/* driver function*/
int main()
{
    int n = 5, m = 4;
    cout << counTriangles(n, m);
    return 0;
}
```

Java

```
 //Java program to count number of triangles
```

```
// with n total points, out of which m are
// collinear.
import java.io.*;
import java.util.*;

class GFG {

// Returns value of binomial coefficient
// Code taken from https://goo.gl/vhy4jp
static int nCk(int n, int k)
{
    int[] C=new int[k+1];
    for (int i=0;i<=k;i++)
    C[i]=0;

    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal triangle
        // using the previous row
        for (int j = Math.min(i, k); j > 0; j--)
            C[j] = C[j] + C[j-1];
    }
    return C[k];
}

/* function to calculate number of triangle
can be formed */
static int counTriangles(int n,int m)
{
    return (nCk(n, 3) - nCk(m, 3));
}

    public static void main (String[] args) {
      int n = 5, m = 4;
      System.out.println(counTriangles(n, m));

    }
}

//This code is contributed by Gitanjali.
```

Python3

```
 # python program to count number of triangles
# with n total points, out of which m are
# collinear.
```

```
import math

# Returns value of binomial coefficient
# Code taken from https://goo.gl / vhy4jp
def nCk(n, k):
    C = [0 for i in range(0, k + 2)]

    C[0] = 1; # nC0 is 1
    for i in range(0, n + 1):

        # Compute next row of pascal triangle
        # using the previous row
        for j in range(min(i, k), 0, -1):
            C[j] = C[j] + C[j-1]

    return C[k]

# function to calculate number of triangle
# can be formed
def counTriangles(n, m):
    return (nCk(n, 3) - nCk(m, 3))

# driver code
n = 5
m = 4
print (counTriangles(n, m))

# This code is contributed by Gitanjali
```

C#

```
 //C# program to count number of triangles
// with n total points, out of which m are
// collinear.
using System;

class GFG {

    // Returns value of binomial coefficient
    // Code taken from https://goo.gl/vhy4jp
    static int nCk(int n, int k)
    {
        int[] C=new int[k+1];
        for (int i = 0; i <= k; i++)
        C[i] = 0;

        // nC0 is 1
        C[0] = 1;
        for (int i = 1; i <= n; i++)
        {
            // Compute next row of pascal triangle
            // using the previous row
            for (int j = Math.Min(i, k); j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }
        return C[k];
    }

    /* function to calculate number of triangle
    can be formed */
    static int counTriangles(int n,int m)
    {
        return (nCk(n, 3) - nCk(m, 3));
    }

    // Driver code
    public static void Main ()
    {
        int n = 5, m = 4;
        Console.WriteLine(counTriangles(n, m));

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count number
// of triangles with n total
// points, out of which m are collinear.

// Returns value of binomial coefficient
// Code taken from https://goo.gl/vhy4jp
function nCk($n, $k)
{
    for ($i = 0; $i <= $k; $i++)
    $C[$i] = 0;

    $C[0] = 1; // nC0 is 1

    for ($i = 1; $i <= $n; $i++)
    {
        // Compute next row of pascal
        // triangle using the previous row
        for ($j = min($i, $k); $j > 0; $j--)
            $C[$j] = $C[$j] + $C[$j - 1];
    }
    return $C[$k];
}

/* function to calculate number
of triangles that can be formed */
function counTriangles($n, $m)
{
    return (nCk($n, 3) - nCk($m, 3));
}

// Driver Code
$n = 5;
$m = 4;
echo counTriangles($n, $m);
return 0;

// This code is contributed by ChitraNayal
?>
```

Output :

```
6
```

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/count-triangles-total-n-points-m-collinear/

Chapter 27

Count pairs with Bitwise AND as ODD number

Count pairs with Bitwise AND as ODD number - GeeksforGeeks

Given an array of N integers. The task is to find the number of pairs (i, j) such that A[i] & A[j] is odd.

Examples:

> **Input:** N = 4
> A[] = { 5, 1, 3, 2 }
> **Output:** 3
> Since pair of A[] = (5, 1), (5, 3), (5, 2), (1, 3), (1, 2), (3, 2)
> 5 AND 1 = 1, 5 AND 3 = 1, 5 AND 2 = 0,
> 1 AND 3 = 1, 1 AND 2 = 0,
> 3 AND 2 = 2
> Total odd pair A(i, j) = 3
>
> **Input :** N = 6
> A[] = { 5, 9, 0, 6, 7, 3 }
> **Output :**6
> Since pair of A[] =
> (5, 9) = 1, (5, 0) = 0, (5, 6) = 4, (5, 7) = 5, (5, 3) = 1,
> (9, 0) = 0, (9, 6) = 0, (9, 7) = 1, (9, 3) = 1,
> (0, 6) = 0, (0, 7) = 0, (0, 3) = 0,
> (6, 7) = 6, (6, 3) = 2,
> (7, 3) = 3

A **naive approach** is to check for every pair and print the count of pairs.

Below is the implementation of the above approach:

C++

```
 // C++ program to count pairs
// with AND giving a odd number
#include <iostream>
using namespace std;

// Function to count number of odd pairs
int findOddPair(int A[], int N)
{
    int i, j;

    // variable for counting odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++) {
        for (j = i + 1; j < N; j++) {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }
    // return number of odd pair
    return oddPair;
}
// Driver Code
int main()
{

    int a[] = { 5, 1, 3, 2 };
    int n = sizeof(a) / sizeof(a[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(a, n) << endl;

    return 0;
}
```

Java

```
 // Java program to count pairs
// with AND giving a odd number
class solution_1
{

// Function to count
// number of odd pairs
static int findOddPair(int A[],
                       int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void main(String args[])
{
    int a[] = { 5, 1, 3, 2 };
    int n = a.length;

    // calling function findOddPair
    // and print number of odd pair
    System.out.println(findOddPair(a, n));
}
}

// This code is contributed
// by Arnab Kundu
```

C#

```
 // C# program to count pairs
// with AND giving a odd number
using System;
```

```
class GFG
{

// Function to count
// number of odd pairs
static int findOddPair(int []A,
                       int N)
{
    int i, j;

    // variable for counting
    // odd pairs
    int oddPair = 0;

    // find all pairs
    for (i = 0; i < N; i++)
    {
        for (j = i + 1; j < N; j++)
        {

            // find AND operation
            // check odd or even
            if ((A[i] & A[j]) % 2 != 0)
                oddPair++;
        }
    }

    // return number
    // of odd pair
    return oddPair;
}

// Driver Code
public static void Main()
{
    int []a = { 5, 1, 3, 2 };
    int n = a.Length;

    // calling function findOddPair
    // and print number of odd pair
    Console.WriteLine(findOddPair(a, n));
}
}

// This code is contributed
// inder_verma.
```

Python

```
 # Python program to count pairs
# with AND giving a odd number

# Function to count number
# of odd pairs
def findOddPair(A, N):

    # variable for counting odd pairs
    oddPair = 0

    # find all pairs
    for i in range(0, N - 1):
        for j in range(i + 1, N - 1):

            # find AND operation
            # check odd or even
            if ((A[i] & A[j]) % 2 != 0):
                oddPair = oddPair + 1

    # return number of odd pair
    return oddPair

# Driver Code
a = [5, 1, 3, 2]
n = len(a)

# calling function findOddPair
# and print number of odd pair
print(findOddPair(a, n))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to count pairs
// with AND giving a odd number

// Function to count
// number of odd pairs
function findOddPair(&$A, $N)
{

    // variable for counting
    // odd pairs
    $oddPair = 0;
    // find all pairs
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = $i + 1; $j < $N; $j++)
        {

            // find AND operation
            // check odd or even
            if (($A[$i] & $A[$j]) % 2 != 0)
                $oddPair = $oddPair + 1;
        }
    }

    // return number of odd pair
    return $oddPair;
}

// Driver Code
$a = array(5, 1, 3, 2);
$n = sizeof($a);

// calling function findOddPair
// and print number of odd pair
echo(findOddPair($a, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
3
```

Time Complexity:O(N^2)

An **efficient solution** is to count the odd numbers. Then return **count * (count – 1)/2** because AND of two numbers can be odd only if only if a pair of both numbers are odd.

C++

```
 // C++ program to count pairs with Odd AND
#include <iostream>
using namespace std;

int findOddPair(int A[], int N)
{
    // Count total odd numbers in
    int count = 0;
    for (int i = 0; i < N; i++)
        if ((A[i] % 2 == 1))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver main
int main()
{
    int a[] = { 5, 1, 3, 2 };
    int n = sizeof(a) / sizeof(a[0]);

    // calling function findOddPair
    // and print number of odd pair
    cout << findOddPair(a, n) << endl;
    return 0;
}
```

Java

```
 // Java program to count
// pairs with Odd AND
class solution_1
{
static int findOddPair(int A[],
                       int N)
{
    // Count total odd numbers in
    int count = 0;
    for (int i = 0; i < N; i++)
        if ((A[i] % 2 == 1))
            count++;

    // return count of even pair
    return count * (count - 1) / 2;
}

// Driver Code
public static void main(String args[])
{
    int a[] = { 5, 1, 3, 2 };
    int n = a.length;

    // calling function findOddPair
    // and print number of odd pair
    System.out.println(findOddPair(a, n));

}
}

// This code is contributed
// by Arnab Kundu
```

Python

```
 # Python program to count
# pairs with Odd AND
def findOddPair(A, N):

    # Count total odd numbers
    count = 0;
    for i in range(0, N - 1):
        if ((A[i] % 2 == 1)):
            count = count+1

    # return count of even pair
    return count * (count - 1) / 2

# Driver Code
a = [5, 1, 3, 2]
n = len(a)

# calling function findOddPair
# and print number of odd pair
print(int(findOddPair(a, n)))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// PHP program to count
// pairs with Odd AND
function findOddPair(&$A, $N)
{
    // Count total odd numbers
    $count = 0;
    for ($i = 0; $i < $N; $i++)
        if (($A[$i] % 2 == 1))
            $count++;
    // return count of even pair
    return $count * ($count - 1) / 2;
}

// Driver Code
$a = array(5, 1, 3, 2 );
$n = sizeof($a);

// calling function findOddPair
// and print number of odd pair
echo(findOddPair($a, $n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
3
```

Time Complexity: O(N)

Improved By : andrew1234, inderDuMCA, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/count-pairs-with-bitwise-and-as-odd-number/

Chapter 28

Count permutations that produce positive result

Count permutations that produce positive result - GeeksforGeeks

Given an array of digits of length n > 1, digits lies within range 0 to 9. We perform sequence of below three operations until we are done with all digits

1. Select starting two digits and add (+)
2. Then next digit is subtracted (−) from result of above step.
3. The result of above step is multiplied (X) with next digit.

We perform above sequence of operations linearly with remaining digits.

The task is to find how many permutations of given array that produce positive result after above operations.

For Example, consider input number[] = {1, 2, 3, 4, 5}. Let us consider a permutation 21345 to demonstrate sequence of operations.

1. Add first two digits, result = 2+1 = 3
2. Subtract next digit, result=result-3= 3-3 = 0
3. Multiply next digit, result=result*4= 0*4 = 0
4. Add next digit, result = result+5 = 0+5 = 5
5. result = 5 which is positive so increment count by one

Examples:

```
Input : number[]="123"
Output: 4
// here we have all permutations
// 123 --> 1+2 -> 3-3 -> 0
// 132 --> 1+3 -> 4-2 -> 2 ( positive )
// 213 --> 2+1 -> 3-3 -> 0
// 231 --> 2+3 -> 5-1 -> 4 ( positive )
// 312 --> 3+1 -> 4-2 -> 2 ( positive )
// 321 --> 3+2 -> 5-1 -> 4 ( positive )
// total 4 permutations are giving positive result

Input : number[]="112"
Output: 2
// here we have all permutations possible
// 112 --> 1+1 -> 2-2 -> 0
// 121 --> 1+2 -> 3-1 -> 2 ( positive )
// 211 --> 2+1 -> 3-1 -> 2 ( positive )
```

Asked in : Morgan Stanley

We first generate all possible permutations of given digit array and perform given sequence of operations sequentially on each permutation and check for which permutation result is positive. Below code describes problem solution easily.

Note : We can generate all possible permutations either by using iterative method, see this article or we can use STL function next_permutation() function to generate it.

```
 // C++ program to find count of permutations that produce
// positive result.
#include<bits/stdc++.h>
using namespace std;

// function to find all permutation after executing given
// sequence of operations and whose result value is positive
// result > 0 ) number[] is array of digits of length of n
int countPositivePermutations(int number[], int n)
{
    // First sort the array so that we get all permutations
    // one by one using next_permutation.
    sort(number, number+n);

    // Initialize result (count of permutations with positive
    // result)
    int count = 0;

    // Iterate for all permutation possible and do operation
    // sequentially in each permutation
    do
    {
        // Stores result for current permutation. First we
        // have to select first two digits and add them
        int curr_result = number[0] + number[1];
        // flag that tells what operation we are going to
        // perform
        // operation = 0 ---> addition operation ( + )
        // operation = 1 ---> subtraction operation ( - )
        // operation = 0 ---> multiplication operation ( X )
        // first sort the array of digits to generate all
        // permutation in sorted manner
        int operation = 1;

        // traverse all digits
        for (int i=2; i<n; i++)
        {
            // sequentially perform + , - , X operation
            switch (operation)
            {
            case 0:
                curr_result += number[i];
                break;
            case 1:
                curr_result -= number[i];
                break;
            case 2:
                curr_result *= number[i];
                break;
            }

            // next operation (decides case of switch)
            operation = (operation + 1) % 3;
        }

        // result is positive then increment count by one
        if (curr_result > 0)
            count++;

    // generate next greater permutation until it is
    // possible
    } while(next_permutation(number, number+n));

    return count;
}

// Driver program to test the case
int main()
{
    int number[] = {1, 2, 3};
    int n = sizeof(number)/sizeof(number[0]);
    cout << countPositivePermutations(number, n);
    return 0;
}
```

Output:

```
 4
```

If you have better and optimized solution for this problem then please share in comments.

Source

https://www.geeksforgeeks.org/count-permutations-produce-positive-result/

Chapter 29

Count strings with consonants and vowels at alternate position

Count strings with consonants and vowels at alternate position - GeeksforGeeks

Given a string **str**. The task is to find all possible number of strings that can be obtained by replacing the "$" with alphabets in the given string.

Note: Alphabets should be placed in such a way that the string is always alternating in vowels and consonants, and the string must always start with a consonant. It is assumed that such a string is always possible, i.e. there is no need to care about the characters other than "$".

Examples:

```
Input: str = "y$s"
Output: 5
$ can be replaced with any of the 5 vowels.
So, there can be 5 strings.

Input: str = "s$$e$"
Output: 2205
```

Approach: It is given that the string will start with a consonant. So, if '$' is at even position(considering 0-based indexing) then there should be a consonant else there should be a vowel. Also, given that there is no need to care about the characters other than "$", i.e., characters other than "$" are placed correctly in the string maintaining the alternating consonant and vowel sequence. Let us understand the problem with an example.

> **str = "s$$e$"**
> Here we have to find the number of ways to form a string with given constraints.
>
> - First occurence of $ is at 2nd position i.e. 1st index, so we can use 5 vowels.

- Second occurence of $ is at 3rd position, so we can use 21 consonants.
- Third occurence of $ is at 5th position, so we can use 21 consonants.

So, total number of ways to form above string is = **5*21*21 = 2205**

Below is the implementation of the above approach:

```
 // C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the count of strings
int countStrings(string s)
{
    // Variable to store the final result
    long sum = 1;

    // Loop iterating through string
    for (int i = 0; i < s.size(); i++) {

        // If '$' is present at the even
        // position in the string
        if (i % 2 == 0 && s[i] == '$')

            //'sum' is multiplied by 21
            sum *= 21;

        // If '$' is present at the odd
        // position in the string
        else if (s[i] == '$')

            //'sum' is multiplied by 5
            sum *= 5;
    }

    return sum;
}

// Driver code
int main()
{
    // Let the string 'str' be s$$e$
    string str = "s$$e$";

    // Print result
    cout << countStrings(str) << endl;
    return 0;
}
```

Output:

```
2205
```

Source

https://www.geeksforgeeks.org/count-strings-with-consonants-and-vowels-at-alternate-position/

Chapter 30

Count unique subsequences of length K

Count unique subsequences of length K - GeeksforGeeks

Given an array of N numbers and an integer K. The task is to print the number of unique subsequences possible of length K.

Examples:

```
Input : a[] = {1, 2, 3, 4}, k = 3
Output : 4.
Unique Subsequences are:
{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}

Input: a[] = {1, 1, 1, 2, 2, 2 }, k = 3
Output : 4
Unique Subsequences are
{1, 1, 1}, {1, 1, 2}, {1, 2, 2}, {2, 2, 2}
```

Approach: There is a well-known formula how many subsequences of fixed length K can be chosen from N unique objects. But the problem here has several differences. One among them is the order in subsequences is important and must be preserved as in the original sequence. For such a problem there can be no ready combinatorics formula because the results depend on the order of the original array.
The main idea is to deal recurrently by the length of the subsequence. On each recurrent step, move from the end to the beginning and count the unique combinations using the count of shorter unique combinations from the previous step. More strictly on every step j we keep an array of length N and every element in the place p means how many unique subsequences with length j we found to the right of the element in place i, including i itself.

Below is the implementation of the above approach.

```
 #include <bits/stdc++.h>
using namespace std;

// Function which returns the numbe of
// unique subsequences of length K
int solution(vector<int>& A, int k)
{
    // seiz of the vector
    // which does is constant
    const int N = A.size();

    // bases cases
    if (N < k || N < 1 || k < 1)
        return 0;
    if (N == k)
        return 1;

    // Prepare arrays for recursion
    vector<int> v1(N, 0);
    vector<int> v2(N, 0);
    vector<int> v3(N, 0);

    // initiate separately for k = 1
    // intiate the last element
    v2[N - 1] = 1;
    v3[A[N - 1] - 1] = 1;

    // initiate all other elements of k = 1
    for (int i = N - 2; i >= 0; i--) {

        // initialize the front element
        // to vector v2
        v2[i] = v2[i + 1];

        // if element v[a[i]-1] is 0
        // then increment it in vector v2
        if (v3[A[i] - 1] == 0) {
            v2[i]++;
            v3[A[i] - 1] = 1;
        }
    }

    // iterate for all possible values of K
    for (int j = 1; j < k; j++) {

        // fill the vectors with 0
        fill(v3.begin(), v3.end(), 0);
        // fill(v1.begin(), v1.end(), 0)
        // the last must be 0 as from last no unique
        // subarray can be formed
        v1[N - 1] = 0;

        // Iterate for all index from which unique
        // subsequences can be formed
        for (int i = N - 2; i >= 0; i--) {

            // add the number of subsequence formed
            // from the next index
            v1[i] = v1[i + 1];

            // start with combinations on the
            // next index
            v1[i] = v1[i] + v2[i + 1];

            // Remove the elements which have
            // already been counted
            v1[i] = v1[i] - v3[A[i] - 1];

            // Update the number used
            v3[A[i] - 1] = v2[i + 1];
        }

        // prepare the next iteration
        // by filling v2 in v1
        v2 = v1;
    }

    // last answer is stored in v1
    return v1[0];
}

// Function to push the vector into an array
// and print all the unique subarrays
void solve(int a[], int n, int k)
{
    vector<int> v;

    // fill the vector with a[]
    v.assign(a, a + n);

    // Function call to print the count
    // of unique susequences of size K
    cout << solution(v, k);
}
```

```
// Driver Code
int main()
{
    int a[] = { 1, 2, 3, 4 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
    solve(a, n, k);

    return 0;
}
```

Output:

```
4
```

Source

https://www.geeksforgeeks.org/count-unique-subsequences-of-length-k/

Chapter 31

Count ways to distribute m items among n people

Count ways to distribute m items among n people - GeeksforGeeks

Given m and n representing number of mangoes and number of people respectively. Task is to calculate number of ways to distribute m mangoes among n people. Considering both variables m and n, we arrive at 4 typical use cases where mangoes and people are considered to be:

1) Both identical
2) Unique and identical respectively
3) Identical and unique respectively
4) Both unique

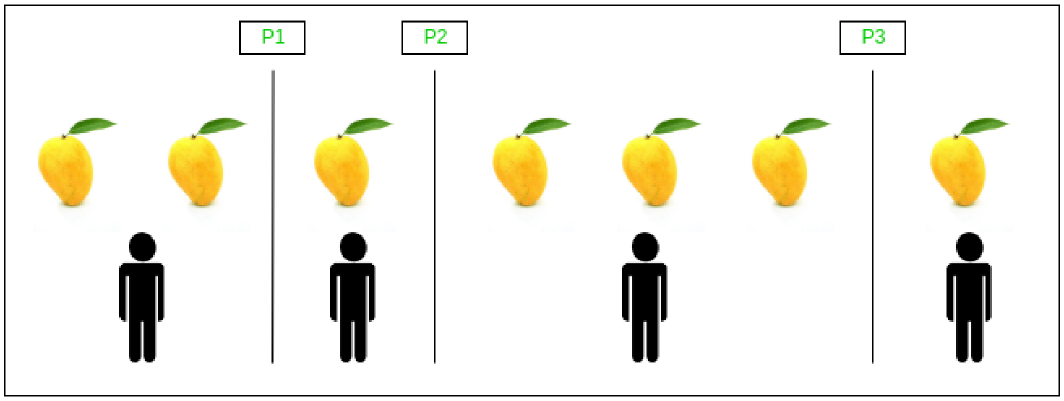
Prerequisites: Binomial Coefficient | Permutation and Combination

Case 1: Distributing m identical mangoes amongst n identical people

If we try to spread m mangoes in a row, our goal is to divide these m mangoes among n people sitting somewhere between arrangement of these mangoes. All we need to do is pool these m mangoes into n sets so that each of these n sets can be allocated to n people respectively.

To accomplish above task, we need to partition the initial arrangement of mangoes by using n-1 partitioners to create n sets of mangoes. In this case we need to arrange m mangoes and n-1 partitioners all together. So we need [illegible] ways to calculate our answer.

Illustration given below represents an example(a way) of an arrangement of partitions created after placing 3 partitioners namely P1, P2, P3 which partitioned all 7 mangoes into 4 different partitions so that 4 people can have their own portion of respective partition:



As all the mangoes are considered to be identical, we divide $(m + n - 1)!$ by $(m)!$ to deduct the duplicate entries. Similarly we divide the above expression again by $(n - 1)!$ because all people are considered to be identical too.

The final expression we get is : [illegible]

The above expression is even-actually equal to the binomial coefficient: [illegible]

Example:

```
Input :  m = 3, n = 2
Output : 4
There are four ways
3 + 0, 1 + 2, 2 + 1 and 0 + 3

Input :  m = 13, n = 6
Output : 8568

Input :  m = 11, n = 3
Output : 78
```

C++

```
 // C++ code for calculating number of ways
// to distribute m mangoes amongst n people
// where all mangoes and people are identical
#include <bits/stdc++.h>
using namespace std;

// function used to generate binomial coefficient
// time complexity O(m)
int binomial_coefficient(int n, int m)
{
    int res = 1;

    if (m > n - m)
        m = n - m;

    for (int i = 0; i < m; ++i) {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}

// helper function for generating no of ways
// to distribute m mangoes amongst n people
int calculate_ways(int m, int n)
{
    // not enough mangoes to be distributed
    if (m < n)
        return 0;

    // ways  -> (n+m-1)C(n-1)
    int ways = binomial_coefficient(n + m - 1, n - 1);
    return ways;
}

// Driver function
int main()
{
    // m represents number of mangoes
    // n represents number of people
    int m = 7, n = 5;

    int result = calculate_ways(m, n);
    printf("%d\n", result);

    return 0;
}
```

Java

```
 // Java code for calculating number of ways
// to distribute m mangoes amongst n people
// where all mangoes and people are identical
```

```
import java.util.*;

class GFG {

    // function used to generate binomial coefficient
    // time complexity O(m)
    public static int binomial_coefficient(int n, int m)
    {
        int res = 1;

        if (m > n - m)
            m = n - m;

        for (int i = 0; i < m; ++i) {
            res *= (n - i);
            res /= (i + 1);
        }

        return res;
    }

    // helper function for generating no of ways
    // to distribute m mangoes amongst n people
    public static int calculate_ways(int m, int n)
    {

        // not enough mangoes to be distributed
        if (m < n) {
            return 0;
        }

        // ways  -> (n+m-1)C(n-1)
        int ways = binomial_coefficient(n + m - 1, n - 1);
        return ways;
    }

    // Driver function
    public static void main(String[] args)
    {

        // m represents number of mangoes
        // n represents number of people
        int m = 7, n = 5;

        int result = calculate_ways(m, n);
        System.out.println(Integer.toString(result));

        System.exit(0);
    }
}
```

Python3

```
 # Python code for calculating number of ways
# to distribute m mangoes amongst n people
# where all mangoes and people are identical


# function used to generate binomial coefficient
# time complexity O(m)
def binomial_coefficient(n, m):
    res = 1

    if m > n - m:
        m = n - m

    for i in range(0, m):
        res *= (n - i)
        res /= (i + 1)

    return res

# helper function for generating no of ways
# to distribute m mangoes amongst n people
def calculate_ways(m, n):

    # not enough mangoes to be distributed
    if m<n:
        return 0

    # ways  -> (n + m-1)C(n-1)
    ways = binomial_coefficient(n + m-1, n-1)
    return int(ways)

# Driver function
if __name__ == '__main__':

    # m represents number of mangoes
    # n represents number of people
    m = 7 ;n = 5

    result = calculate_ways(m, n)
    print(result)
```

C#

```
 // C# code for calculating number
// of ways to distribute m mangoes
// amongst n people where all mangoes
// and people are identical
using System;

class GFG
{

// function used to generate
// binomial coefficient
// time complexity O(m)
public static int binomial_coefficient(int n,
                                       int m)
{
    int res = 1;

    if (m > n - m)
        m = n - m;

    for (int i = 0; i < m; ++i)
    {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}

// helper function for generating
// no of ways to distribute m
// mangoes amongst n people
public static int calculate_ways(int m, int n)
{

    // not enough mangoes
    // to be distributed
    if (m < n)
    {
        return 0;
    }

    // ways -> (n+m-1)C(n-1)
    int ways = binomial_coefficient(n + m - 1,
                                    n - 1);
    return ways;
}
// Driver Code
public static void Main()
{

    // m represents number of mangoes
    // n represents number of people
    int m = 7, n = 5;

    int result = calculate_ways(m, n);
    Console.WriteLine(result.ToString());
}
}

// This code is contributed
// by Subhadeep
```

PHP

```
 <?php
// PHP code for calculating number
// of ways to distribute m mangoes
// amongst n people where all
// mangoes and people are identical

// function used to generate
// binomial coefficient
// time complexity O(m)
function binomial_coefficient($n, $m)
{
    $res = 1;

    if ($m > $n - $m)
        $m = $n - $m;

    for ($i = 0; $i < $m; ++$i)
    {
        $res *= ($n - $i);
        $res /= ($i + 1);
    }

    return $res;
}

// Helper function for generating
// no of ways to distribute m.
// mangoes amongst n people
function calculate_ways($m, $n)
{
    // not enough mangoes to
    // be distributed
    if ($m < $n)
        return 0;

    // ways -> (n+m-1)C(n-1)
    $ways = binomial_coefficient($n + $m - 1,
                                  $n - 1);
    return $ways;
}

// Driver Code

// m represents number of mangoes
// n represents number of people
$m = 7;
$n = 5;

$result = calculate_ways($m, $n);
echo $result;

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
330
```

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

Case 2: Distributing m unique mangoes amongst n identical people
In this case, to calculate the number of ways to distribute m unique mangoes amongst n identical people, we just need to multiply the last expression [illegible] we calculated in Case 1 by $m!$.

So our final expression for this case is [illegible]

Proof:

In case 1, initially we got the expression [illegible] without removing duplicate entries.

In this case, we only need to divide [illegible] as all mangoes are considered to be unique in this case.

So we get the expression as : [illegible]

Multiplying both numerator and denominator by $(n-1)!$,

we get [illegible]

Where [illegible] === [illegible]

Time Complexity : O(max(n, m))
Auxiliary Space : O(1)

Case 3: Distributing m identical mangoes amongst n unique people

In this case, to calculate the number of ways to distribute m identical mangoes amongst n unique people, we just need to multiply the last expression [illegible] we calculated in Case 1 by $(n-1)!$.

So our final expression for this case is [illegible] $C_{n-1} * (n-1)!$

Proof:
This Proof is pretty much similar to the proof of last case expression.

In case 1, initially we got the expression [illegible] without removing duplicate entries.

In this case, we only need to divide $n!$ as all people are considered to be unique in this case.

So we get the expression as : [illegible]

Multiplying both numerator and denominator by $(n-1)!$,

we get [illegible]

Where [illegible] ===
[illegible] $C_{n-1} * (n-1)!$

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

For references on how to calculate $n!$ refer here factorial of a number

Case 4: Distributing m unique mangoes amongst n unique people

In this case we need to multiply the expression obtained in case 1 by both $n!$ and $(n-1)!$.

The proofs for both of the multiplications are defined in case 2 and case 3.

Hence, in this case, our final expression comes out to be [illegible] $C_{n-1} * (n-1)! * m!$

Time Complexity : O(n+m)
Auxiliary Space : O(1)

Improved By : tufan_gupta2000, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/count-ways-to-distribute-m-items-among-n-people/

Chapter 32

Count ways to express even number 'n' as sum of even integers

Count ways to express even number 'n' as sum of even integers - GeeksforGeeks

Given an positive even integer 'n'. Count total number of ways to express 'n' as sum of even positive integers. Output the answer in modulo $\mathbf{10^9 + 7}$

Examples:

```
Input: 6
Output: 4

Explanation
There are only four ways to write 6
as sum of even integers:
1. 2 + 2 + 2
2. 2 + 4
3. 4 + 2
4. 6
Input: 8
Output: 8
```

Approach is to find pattern or recursive function whichever is possible. The approach would be the same as already discussed in "Count ways to express 'n' as sum of odd integers". Here the given number is even that means even sum can only be achieved by adding the (n-2)$^{\text{th}}$ number as two times. We can notice that (by taking some examples) adding a 2 to a number doubles the count. Let the total number of ways to write 'n' be ways(n). The value of 'ways(n)' can be written by formula as follows:

```
ways(n) = ways(n-2) + ways(n-2)
ways(n) = 2 * ways(n-2)

ways(2) = 1 = 20
ways(4) = 2 = 21
ways(6) = 4 = 22
ways(8) = 8 = 23
''
''
''
ways(2 * n) = 2n-1

Replace n by (m / 2)
=> ways(m) = 2m/2 - 1
```

C++

```
 // C++ program to count ways to write
// number as sum of even integers
#include<iostream>
using namespace std;

// Initialize mod variable as constant
const int MOD = 1e9 + 7;

/* Iterative Function to calculate (x^y)%p in O(log y) */
int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result

    x = x % p;  // Update x if it is more than or
                // equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (1LL * res * x) % p;

        // y must be even now
        y = y>>1; // y = y/2
        x = (1LL * x * x) % p;
    }
    return res;
}

// Return number of ways to write 'n'
// as sum of even integers
int countEvenWays(int n)
{
  return power(2, n/2 - 1, MOD);
}

// Driver code
int main()
{
    int n = 6;
    cout << countEvenWays(n) << "\n";

    n = 8;
    cout << countEvenWays(n);
   return 0;
}
```

Java

```
 // JAVA program to count ways to write
// number as sum of even integers

class GFG {

    // Initialize mod variable as constant
    static int MOD = 1000000007;

    /* Iterative Function to calculate
    (x^y)%p in O(log y) */
    static int power(int x, int y, int p)
    {
        // Initialize result
        int res = 1;

        // Update x if it is more
        // than or equal to p
        x = x % p;

        while (y > 0)
        {
            // If y is odd, multiply x
            // with result
            if (y % 2 == 1)
                res = (1 * res * x) % p;

            // y must be even now
            y = y >> 1; // y = y/2
            x = (1 * x * x) % p;
        }
        return res;
    }

    // Return number of ways to write
    // 'n' as sum of even integers
    static int countEvenWays(int n)
    {
      return power(2, n/2 - 1, MOD);
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 6;
        System.out.println(countEvenWays(n));
        n = 8;
        System.out.println(countEvenWays(n));
    }
}

/* This code is contributed by Nikita Tiwari. */
```

Python

```
 # PYTHON program to count ways to write
# number as sum of even integers

# Initialize mod variable as constant
MOD = 1e9 + 7

# Iterative Function to calculate
# (x^y)%p in O(log y)
def power(x, y, p) :
    res = 1      # Initialize result

    x = x % p  # Update x if it is more
               # than or equal to p

    while (y > 0) :

        # If y is odd, multiply x
        # with result
        if (y & 1) :
            res = (1 * res * x) % p

        # y must be even now
        y = y >> 1  # y = y/2
        x = (1 * x * x) % p


    return res


# Return number of ways to write 'n'
# as sum of even integers
def countEvenWays(n) :
    return power(2, n/2 - 1, MOD)

# Driver code
n = 6
print (int(countEvenWays(n)))
n = 8
print (int(countEvenWays(n)))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to count ways to write
// number as sum of even integers
using System;

class GFG {

    // Initialize mod variable as constant
    static int MOD = 1000000007;

    /* Iterative Function to calculate
    (x^y)%p in O(log y) */
    static int power(int x, int y, int p)
    {

        // Initialize result
        int res = 1;

        // Update x if it is more
        // than or equal to p
        x = x % p;

        while (y > 0)
        {

            // If y is odd, multiply x
            // with result
            if (y % 2 == 1)
```

```
                res = (1 * res * x) % p;

            // y must be even now
            y = y >> 1; // y = y/2
            x = (1 * x * x) % p;
        }

        return res;
    }

    // Return number of ways to write
    // 'n' as sum of even integers
    static int countEvenWays(int n)
    {
        return power(2, n/2 - 1, MOD);
    }

    // Driver code
    public static void Main()
    {
        int n = 6;
        Console.WriteLine(countEvenWays(n));

        n = 8;
        Console.WriteLine(countEvenWays(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count ways
// to write number as sum of
// even integers

// Initialize mod variable
// as constant
$MOD = 1000000000.0;

/* Iterative Function to
calculate (x^y)%p in O(log y) */
function power($x, $y, $p)
{
    // Initialize result
    $res = 1;
    // Update x if it is more
    // than or equal to p
    $x = $x % $p;


    while ($y > 0)
    {
        // If y is odd, multiply
        // x with result
        if ($y & 1)
            $res = (1 * $res *
                        $x) % $p;

        // y must be even now
        $y = $y >> 1; // y = y/2
        $x = (1 * $x *
                  $x) % $p;
    }
    return $res;
}

// Return number of ways
// to write 'n' as sum of
// even integers
function countEvenWays($n)
{
    global $MOD;
    return power(2, $n /
                 2 - 1, $MOD);
}

// Driver code
$n = 6;
echo countEvenWays($n), "\n";

$n = 8;
echo countEvenWays($n);

// This code is contributed
// by ajit
?>
```

Output:

```
4
8
```

Time complexity: O(Log(n))
Auxiliary space: O(1)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/count-ways-express-even-number-n-sum-even-integers/

Chapter 33

Count ways to express ‘n’ as sum of odd integers

Count ways to express ’n’ as sum of odd integers - GeeksforGeeks

Given an positive integer n. Count total number of ways to express ‘n’ as sum of odd positive integers.

```
Input: 4
Output: 3

Explanation
There are only three ways to write 4
as sum of odd integers:
1. 1 + 3
2. 3 + 1
3. 1 + 1 + 1 + 1

Input: 5
Output: 5
```

Simple approach is to find recursive nature of problem. The number ‘n’ can be written as sum of odd integers from either (n-1)th number or (n-2)th number. Let the total number of ways to write ‘n’ be ways(n). The value of ‘ways(n)’ can be written by recursive formula as follows:

```
ways(n) = ways(n-1) + ways(n-2)
```

The above expression is actually the expression for Fibonacci numbers. Therefore problem is reduced to find the nth fibonnaci number.

```
ways(1) = fib(1) = 1
ways(2) = fib(2) = 1
ways(3) = fib(2) = 2
ways(4) = fib(4) = 3
```

C++

```
 // C++ program to count ways to write
// number as sum of odd integers
#include<iostream>
using namespace std;

// Function to calculate n'th Fibonacci number
int fib(int n)
{
  /* Declare an array to store Fibonacci numbers. */
  int f[n+1];
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
      /* Add the previous 2 numbers in the series
         and store it */
      f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}

// Return number of ways to write 'n'
// as sum of odd integers
int countOddWays(int n)
{
    return fib(n);
}

// Driver code
int main()
{
    int n = 4;
    cout << countOddWays(n) << "\n";

    n = 5;
    cout << countOddWays(n);
   return 0;
}
```

Java

```
 // Java program to count ways to write
// number as sum of odd integers
import java.util.*;

class GFG {

// Function to calculate n'th Fibonacci number
static int fib(int n) {

    /* Declare an array to store Fibonacci numbers. */
    int f[] = new int[n + 1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++) {

    /* Add the previous 2 numbers in the series
        and store it */
    f[i] = f[i - 1] + f[i - 2];
    }

    return f[n];
}

// Return number of ways to write 'n'
// as sum of odd integers
static int countOddWays(int n)
{
    return fib(n);
}

// Driver code
public static void main(String[] args) {

    int n = 4;
    System.out.print(countOddWays(n) + "\n");

    n = 5;
    System.out.print(countOddWays(n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python code to count ways to write
# number as sum of odd integers

# Function to calculate n'th
# Fibonacci number
def fib( n ):

    # Declare a list to store
    # Fibonacci numbers.
    f=list()

    # 0th and 1st number of the
    # series are 0 and 1
    f.append(0)
    f.append(1)

    i = 2
    while i<n+1:

        # Add the previous 2 numbers
        # in the series and store it
        f.append(f[i-1] + f[i-2])
        i += 1
    return f[n]

# Return number of ways to write 'n'
# as sum of odd integers
def countOddWays( n ):
    return fib(n)

# Driver code
n = 4
print(countOddWays(n))
n = 5
print(countOddWays(n))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
 // C# program to count ways to write
```

```
// number as sum of odd integers
using System;

class GFG {

    // Function to calculate n'th
    // Fibonacci number
    static int fib(int n) {

        /* Declare an array to store
        Fibonacci numbers. */
        int []f = new int[n + 1];
        int i;

        /* 0th and 1st number of the
        series are 0 and 1*/
        f[0] = 0;
        f[1] = 1;

        for (i = 2; i <= n; i++)
        {

            /* Add the previous 2 numbers
            in the series and store it */
            f[i] = f[i - 1] + f[i - 2];
        }

        return f[n];
    }

    // Return number of ways to write 'n'
    // as sum of odd integers
    static int countOddWays(int n)
    {
        return fib(n);
    }

    // Driver code
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(countOddWays(n));

        n = 5;
        Console.WriteLine(countOddWays(n));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count ways to write
// number as sum of odd integers

// Function to calculate n'th
// Fibonacci number
function fib($n)
{

    // Declare an array to
    // store Fibonacci numbers.
    $f = array();
    $i;

    // 0th and 1st number of the
    // series are 0 and 1
    $f[0] = 0;
    $f[1] = 1;

    for($i = 2; $i <= $n; $i++)
    {

        // Add the previous 2
        // numbers in the series
        // and store it
        $f[$i] = $f[$i - 1] +
                 $f[$i - 2];
    }

    return $f[$n];
}

// Return number of ways to write 'n'
// as sum of odd integers
function countOddWays( $n)
{
    return fib($n);
}

    // Driver Code
    $n = 4;
    echo countOddWays($n) , "\n";
    $n = 5;
    echo countOddWays($n);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
3
5
```

Note: *The time complexity of the above implementation is O(n). It can be further optimized up-to O(Logn) time using Fibonacci function optimization by Matrix Exponential.*

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-ways-express-n-sum-odd-integers/

Chapter 34

Count ways to form minimum product triplets

Count ways to form minimum product triplets - GeeksforGeeks

Given an array of positive integers. We need to find how many triples of indices (i, j, k) (i < j < k), such that a[i] * a[j] * a[k] is minimum possible.

```
Examples:
Input : 5
        1 3 2 3 4
Output : 2
The triplets are (1, 3, 2)
and (1, 2, 3)

Input : 5
        2 2 2 2 2
Output : 5
In this example we choose three 2s
out of five, and the number of ways
to choose them is 5C3.

Input : 6
        1 3 3 1 3 2
Output : 1
There is only one way (1, 1, 2).
```

Following cases arise in this problem.

1. All three minimum elements are same. For example {1, 1, 1, 1, 2, 3, 4}. The solution for such cases is ${}^{n}C_3$.

2. Two elements are same. For example {1, 2, 2, 2, 3} or {1, 1, 2, 2}. In this case, count of occurrences of first (or minimum element) cannot be more than 2. If minimuum element appears two times, then answer is count of second element (We get to choose only 1 from all occurrences of second element. If minimum element appears once, the count is $^nC_2$.
3. All three elements are distinct. For example {1, 2, 3, 3, 5}. In this case, answer is count of occurrences of third element (or $^nC_1$).

We first sort the array in increasing order. Then count the frequency of 3 element of 3rd element from starting. Let the frequency be 'count'. Following cases arise.

- If 3rd element is equal to the first element, no. of triples will be (count-2)*(count-1)*(count)/6, where count is the frequency of 3rd element.
- If 3rd element is equal to 2nd element, no. of triples will be (count-1)*(count)/2. Otherwise no. of triples will be value of count.

C++

```
 // CPP program to count number of ways we can
// form triplets with minimum product.
#include <bits/stdc++.h>
using namespace std;

// function to calculate number of triples
void noOfTriples(long long arr[], int n)
{
    // Sort the array
    sort(arr, arr + n);

    // Count occurrences of third element
    long long count = 0;
    for (long long i = 0; i < n; i++)
        if (arr[i] == arr[2])
            count++;

    // If all three elements are same (minimum
    // element appears at least 3 times). Answer
    // is nC3.
    if (arr[0] == arr[2])
        return (count - 2) * (count - 1) * (count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if (arr[1] == arr[2])
        return (count - 1) * (count) / 2;

    // Minimum two elements are distinct.
    // Answer is nC1.
    return count;
}

// Driver code
int main()
{
    long long arr[] = { 1, 3, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << noOfTriples(arr, n);
    return 0;
}
```

Java

```
 // Java program to count number of ways we can
// form triplets with minimum product.
import java.util.Arrays;

class GFG {

    // function to calculate number of triples
    static long noOfTriples(long arr[], int n)
    {

        // Sort the array
        Arrays.sort(arr);

        // Count occurrences of third element
        long count = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] == arr[2])
                count++;

        // If all three elements are same (minimum
        // element appears at least 3 times). Answer
        // is nC3.
        if (arr[0] == arr[2])
            return (count - 2) * (count - 1) *
                                  (count) / 6;

        // If minimum element appears once.
        // Answer is nC2.
        else if (arr[1] == arr[2])
            return (count - 1) * (count) / 2;

        // Minimum two elements are distinct.
        // Answer is nC1.
        return count;
    }

    //driver code
    public static void main(String arg[])
    {

        long arr[] = { 1, 3, 3, 4 };
        int n = arr.length;

        System.out.print(noOfTriples(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to count number
# of ways we can form triplets
# with minimum product.

# function to calculate number of triples
def noOfTriples (arr, n):

    # Sort the array
    arr.sort()

    # Count occurrences of third element
    count = 0
    for i in range(n):
        if arr[i] == arr[2]:
            count+=1

    # If all three elements are same
    # (minimum element appears at l
    # east 3 times). Answer is nC3.
    if arr[0] == arr[2]:
        return (count - 2) * (count - 1) * (count) / 6

    # If minimum element appears once.
    # Answer is nC2.
    elif arr[1] == arr[2]:
        return (count - 1) * (count) / 2

    # Minimum two elements are distinct.
    # Answer is nC1.
    return count
```

```
# Driver code
arr = [1, 3, 3, 4]
n = len(arr)
print (noOfTriples(arr, n))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to count number of ways
// we can form triplets with minimum product.
using System;

class GFG {

// function to calculate number of triples
static long noOfTriples(long []arr, int n)
{
    // Sort the array
    Array.Sort(arr);

    // Count occurrences of third element
    long count = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] == arr[2])
            count++;

    // If all three elements are same (minimum
    // element appears at least 3 times). Answer
    // is nC3.
    if (arr[0] == arr[2])
        return (count - 2) * (count - 1) * (count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if (arr[1] == arr[2])
        return (count - 1) * (count) / 2;

    // Minimum two elements are distinct.
    // Answer is nC1.
    return count;
}

//driver code
public static void Main()
{
    long []arr = { 1, 3, 3, 4 };
    int n = arr.Length;
    Console.Write(noOfTriples(arr, n));
}
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to count number of ways
// we can form triplets with minimum
// product.

// function to calculate number of
// triples
function noOfTriples( $arr, $n)
{
    // Sort the array
    sort($arr);

    // Count occurrences of third element
    $count = 0;
    for ( $i = 0; $i < $n; $i++)
        if ($arr[$i] == $arr[2])
            $count++;

    // If all three elements are same
    // (minimum element appears at least
    // 3 times). Answer is nC3.
    if ($arr[0] == $arr[2])
        return ($count - 2) * ($count - 1)
                           * ($count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if ($arr[1] == $arr[2])
        return ($count - 1) * ($count) / 2;

    // Minimum two elements are distinct.
    // Answer is nC1.
    return $count;
}

// Driver code
    $arr = array( 1, 3, 3, 4 );
    $n = count($arr);
    echo noOfTriples($arr, $n);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
1
```

Time Complexity: O(n Log n)

The solution can be optimized by first finding minimum element and its frequency and if frequency is less than 3, then finding second minimum and its frequency. If overall frequency is less than 3, then finding third minimum and its frequency. Time complexity of this optimized solution would be O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-ways-form-minimum-product-triplets/

Chapter 35

Count ways to reach the nth stair using step 1, 2 or 3

Count ways to reach the nth stair using step 1, 2 or 3 - GeeksforGeeks

A child is running up a staircase with n steps and can hop either 1 step, 2 steps, or 3 steps at a time. Implement a method to count how many possible ways the child can run up the stairs.

There are two methods to solve this problem
1. Recursive Method
2. Dynamic Programming

Examples :

```
Input : 4
Output : 7

Input : 3
Output : 4
```

1. Recursive Method
How Code is Working :
Suppose you have n stairs then you can hop either 1 step, 2 step, 3 step.
1. If you hop 1 step then remaining stairs = n-1
2. If you hop 2 step then remaining stairs = n-2
3. If you hop 3 step then remaining stairs = n-3

If you hop 1 step then again you can hop 1 step, 2 step, 3 step until n equals 0.
Repeat this process and count total number of ways to reach at nth stair using step 1, 2, 3.

C

```
 // Program to find n-th stair using step size
// 1 or 2 or 3.
#include <stdio.h>

// Returns count of ways to reach n-th stair
// using 1 or 2 or 3 steps.
int findStep(int n)
{
    if (n == 1 || n == 0)
        return 1;
    else if (n == 2)
        return 2;

    else
        return findStep(n - 3) +
               findStep(n - 2) +
               findStep(n - 1);
}

// Driver code
int main()
{
    int n = 4;
    printf("%d\n", findStep(n));
    return 0;
}
```

Java

```
 // Program to find n-th stair
// using step size 1 or 2 or 3.
import java.util.*;
import java.lang.*;

public class GfG{

    // Returns count of ways to reach
    // n-th stair using 1 or 2 or 3 steps.
    public static int findStep(int n)
    {
        if (n == 1 || n == 0)
            return 1;
        else if (n == 2)
            return 2;

        else
            return findStep(n - 3) +
                   findStep(n - 2) +
                findStep(n - 1);
    }

    // Driver function
    public static void main(String argc[]){
        int n = 4;
        System.out.println(findStep(n));
    }
}

/* This code is contributed by Sagar Shukla */
```

Python

```
 # Python program to find n-th stair
# using step size 1 or 2 or 3.

# Returns count of ways to reach n-th
# stair using 1 or 2 or 3 steps.
def findStep( n) :
    if (n == 1 or n == 0) :
        return 1
    elif (n == 2) :
        return 2

    else :
        return findStep(n - 3) + findStep(n - 2) + findStep(n - 1)


# Driver code
n = 4
print(findStep(n))

#This code is contributed by Nikita Tiwari.
```

C#

```
 // Program to find n-th stair
// using step size 1 or 2 or 3.
using System;

public class GfG{

    // Returns count of ways to reach
    // n-th stair using 1 or 2 or 3 steps.
    public static int findStep(int n)
    {
        if (n == 1 || n == 0)
            return 1;
        else if (n == 2)
            return 2;

        else
            return findStep(n - 3) +
                   findStep(n - 2) +
                   findStep(n - 1);
    }

    // Driver function
    public static void Main(){
        int n = 4;
        Console.WriteLine(findStep(n));
    }
}

/* This code is contributed by vt_m */
```

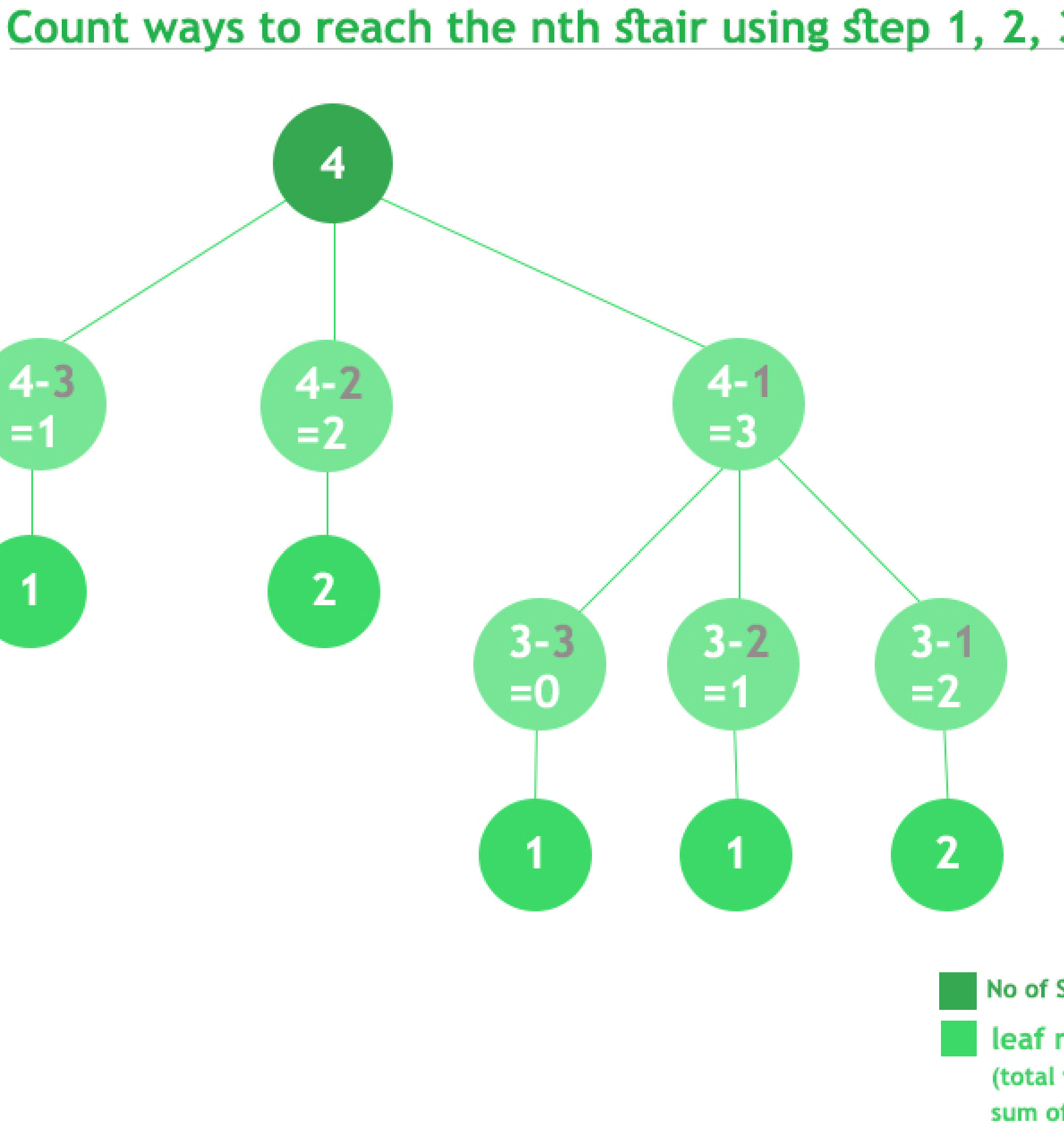
PHP

```
 <?php
// PHP Program to find n-th stair
// using step size 1 or 2 or 3.

// Returns count of ways to
// reach n-th stair using
// 1 or 2 or 3 steps.
function findStep($n)
{
    if ($n == 1 || $n == 0)
        return 1;
    else if ($n == 2)
        return 2;

    else
        return findStep($n - 3) +
               findStep($n - 2) +
                findStep($n - 1);
}

// Driver code
$n = 4;
echo findStep($n);

// This code is contributed by m_kit
?>
```

Output :

```
7
```

How Code is Working....



The Time Complexity of the program is Optimized using Dynamic Programming.

2. Using Dynamic Programming

C

```
 // A C program to count number of ways to reach n't stair when
#include <stdio.h>

// A recursive function used by countWays
int countWays(int n)
{
    int res[n + 1];
    res[0] = 1;
    res[1] = 1;
    res[2] = 2;
    for (int i = 3; i <= n; i++)
        res[i] = res[i-1] + res[i-2]
                  + res[i-3];

    return res[n];
}

// Driver program to test above functions
int main()
{
    int n = 4;
    printf("%d", countWays(n));
    return 0;
}
```

Java

```
 // Program to find n-th stair
// using step size 1 or 2 or 3.
import java.util.*;
import java.lang.*;

public class GfG {

    // A recursive function used by countWays
    public static int countWays(int n)
    {
        int[] res = new int[n + 1];
        res[0] = 1;
        res[1] = 1;
        res[2] = 2;
        for (int i = 3; i <= n; i++)
            res[i] = res[i - 1] + res[i - 2]
                                + res[i - 3];

        return res[n];
    }

    // Driver function
    public static void main(String argc[])
    {
        int n = 4;
        System.out.println(countWays(n));
    }
}

/* This code is contributed by Sagar Shukla */
```

Python

```
 # Python program to find n-th stair
# using step size 1 or 2 or 3.

# A recursive function used by countWays
def countWays(n) :
    res = [0] * (n + 1)
    res[0] = 1
    res[1] = 1
    res[2] = 2

    for i in range(3, n + 1) :
        res[i] = res[i - 1] + res[i - 2] + res[i - 3]

    return res[n]

# Driver code
n = 4
print(countWays(n))


# This code is contributed by Nikita Tiwari.
```

C#

```
 // Program to find n-th stair
// using step size 1 or 2 or 3.
using System;
```

```
public class GfG {

    // A recursive function used by countWays
    public static int countWays(int n)
    {
        int[] res = new int[n + 1];
        res[0] = 1;
        res[1] = 1;
        res[2] = 2;

        for (int i = 3; i <= n; i++)
            res[i] = res[i - 1] + res[i - 2]
                                + res[i - 3];

        return res[n];
    }

    // Driver function
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(countWays(n));
    }
}

/* This code is contributed by vt_m */
```

PHP

```
 <?php
// A PHP program to count
// number of ways to reach
// n'th stair when

// A recursive function
// used by countWays
function countWays($n)
{
    $res[0] = 1;
    $res[1] = 1;
    $res[2] = 2;
    for ($i = 3; $i <= $n; $i++)
        $res[$i] = $res[$i - 1] +
                   $res[$i - 2] +
                   $res[$i - 3];

    return $res[$n];
}

// Driver Code
$n = 4;
echo countWays($n);

// This code is contributed by ajit
?>
```

Output :

```
7
```

Output Explanation :
1 -> 1 -> 1 -> 1
1 -> 1 -> 2
1 -> 2 -> 1
1 -> 3
2 -> 1 -> 1
2 -> 2
3 -> 1

So Total ways: 7

Other Related Articles
http://www.geeksforgeeks.org/count-ways-reach-nth-stair/

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-ways-reach-nth-stair-using-step-1-2-3/

Chapter 36

Count ways to spell a number with repeated digits

Count ways to spell a number with repeated digits - GeeksforGeeks

Given a string that contains digits of a number. The number may contain many same continuous digits in it. The task is to count number of ways to spell the number.
For example, consider 8884441100, one can spell it simply as triple eight triple four double two and double zero. One can also spell as double eight, eight, four, double four, two, two, double zero.

Examples :

```
Input :  num = 100
Output : 2
The number 100 has only 2 possibilities,
1) one zero zero
2) one double zero.

Input : num = 11112
Output: 8
1 1 1 1 2, 11 1 1 2, 1 1 11 2, 1 11 1 2
11 11 2, 1 111 2, 111 1 2, 1111 2

Input : num = 8884441100
Output: 64

Input : num = 12345
Output: 1

Input : num = 11111
Output: 16
```

This is a simple problem of permutation and combination. If we take example test case given in the question, 11112. The answer depends on the number of possible combinations of 1111. The number of combinations of "1111" is 2^3 = 8. As our combinations will depend on whether we choose a particular 1 and for "2" there will be only one possibility 2^0 = 1, so answer for "11112" will be 8*1 = 8.
So, the approach is to count the particular continuous digit in string and multiply 2^(count-1) with previous result.

C++

```
 // C++ program to count number of ways we
// can spell a number
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

// Function to calculate all possible spells of
// a number with repeated digits
// num --> string which is favourite number
ll spellsCount(string num)
{
    int n = num.length();

    // final count of total possible spells
    ll result = 1;

    // iterate through complete number
    for (int i=0; i<n; i++)
    {
       // count contiguous frequency of particular
       // digit num[i]
       int count = 1;
       while (i < n-1 && num[i+1] == num[i])
       {
           count++;
           i++;
       }

       // Compute 2^(count-1) and multiply with result
       result = result * pow(2, count-1);
    }
    return result;
}

// Driver program to run the case
int main()
{
    string num = "11112";
    cout << spellsCount(num);
    return 0;
}
```

Java

```
 // Java program to count number of ways we
// can spell a number
class GFG {

    // Function to calculate all possible
    // spells of a number with repeated digits
    // num --> string which is favourite number
    static long spellsCount(String num)
    {

        int n = num.length();

        // final count of total possible spells
        long result = 1;

        // iterate through complete number
        for (int i = 0; i < n; i++) {

            // count contiguous frequency of
            // particular digit num[i]
            int count = 1;

            while (i < n - 1 && num.charAt(i + 1)
                               == num.charAt(i)) {

                count++;
                i++;
            }

            // Compute 2^(count-1) and multiply
            // with result
            result = result *
                     (long)Math.pow(2, count - 1);
        }
        return result;
    }

    public static void main(String[] args)
    {

        String num = "11112";

        System.out.print(spellsCount(num));
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to count number of ways we
// can spell a number
using System;

class GFG {

    // Function to calculate all possible
    // spells of a number with repeated
    // digits num --> string which is
    // favourite number
    static long spellsCount(String num)
    {

        int n = num.Length;

        // final count of total possible
        // spells
        long result = 1;

        // iterate through complete number
        for (int i = 0; i < n; i++)
        {

            // count contiguous frequency of
            // particular digit num[i]
            int count = 1;

            while (i < n - 1 && num[i + 1]
                             == num[i])
            {
                count++;
                i++;
            }

            // Compute 2^(count-1) and multiply
            // with result
            result = result *
                    (long)Math.Pow(2, count - 1);
        }

        return result;
    }

    // Driver code
    public static void Main()
    {

        String num = "11112";

        Console.Write(spellsCount(num));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to count
// number of ways we
// can spell a number

// Function to calculate
// all possible spells of
// a number with repeated
// digits num --> string
// which is favourite number
function spellsCount($num)
{
    $n = strlen($num);

    // final count of total
    // possible spells
    $result = 1;

    // iterate through
    // complete number
    for ($i = 0; $i < $n; $i++)
    {
    // count contiguous frequency
    // of particular digit num[i]
    $count = 1;
    while ($i < $n - 1 &&
           $num[$i + 1] == $num[$i])
    {
        $count++;
        $i++;
    }
    // Compute 2^(count-1) and
    // multiply with result
    $result = $result *
              pow(2, $count - 1);
    }
    return $result;
}

// Driver Code
$num = "11112";
echo spellsCount($num);

// This code is contributed
// by nitin mittal.
?>
```

Output :

```
8
```

If you have another approach to solve this problem then please share.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-ways-spell-number-repeated-digits/

Chapter 37

Counting pairs when a person can form pair with at most one

Counting pairs when a person can form pair with at most one - GeeksforGeeks

Consider a coding competition on geeksforgeeks practice. Now their are **n** distinct participants taking part in the competition. A single participant can make pair with at most one other participant. We need count the number of ways in which **n** participants participating in the coding competition.

Examples :

```
Input : n = 2
Output : 2
2 shows that either both participant
can pair themselves in one way or both
of them can remain single.

Input : n = 3
Output : 4
One way : Three participants remain single
Three More Ways : [(1, 2)(3)], [(1), (2,3)]
and [(1,3)(2)]
```

1) Every participant can either pair with another participant or can remain single.
2) Let us consider **X-th** participant, he can either remain single or
he can pair up with someone from [**1, x-1**].

C++

```
 // Number of ways in which participant can take part.
#include<iostream>
```

```
using namespace std;

int numberOfWays(int x)
{
    // Base condition
    if (x==0 || x==1)
        return 1;

    // A participant can choose to consider
    // (1) Remains single. Number of people
    //     reduce to (x-1)
    // (2) Pairs with one of the (x-1) others.
    //     For every pairing, number of people
    //     reduce to (x-2).
    else
        return numberOfWays(x-1) +
               (x-1)*numberOfWays(x-2);
}

// Driver code
int main()
{
    int x = 3;
    cout << numberOfWays(x) << endl;
    return 0;
}
```

Java

```
 // Number of ways in which
// participant can take part.
import java.io.*;

class GFG {

static int numberOfWays(int x)
{
    // Base condition
    if (x==0 || x==1)
        return 1;

    // A participant can choose to consider
    // (1) Remains single. Number of people
    //     reduce to (x-1)
    // (2) Pairs with one of the (x-1) others.
    //     For every pairing, number of people
    //     reduce to (x-2).
    else
```

```
        return numberOfWays(x-1) +
            (x-1)*numberOfWays(x-2);
}

// Driver code
public static void main (String[] args) {
int x = 3;
System.out.println( numberOfWays(x));

    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python program to find Number of ways
# in which participant can take part.

# Function to calculate number of ways.
def numberOfWays (x):

    # Base condition
    if x == 0 or x == 1:
        return 1

    # A participant can choose to consider
    # (1) Remains single. Number of people
    # reduce to (x-1)
    # (2) Pairs with one of the (x-1) others.
    # For every pairing, number of people
    # reduce to (x-2).
    else:
        return (numberOfWays(x-1) +
              (x-1) * numberOfWays(x-2))

# Driver code
x = 3
print (numberOfWays(x))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
 // Number of ways in which
// participant can take part.
using System;
```

```
class GFG {

    static int numberOfWays(int x)
    {

        // Base condition
        if (x == 0 || x == 1)
            return 1;

        // A participant can choose to
        // consider (1) Remains single.
        // Number of people reduce to
        // (x-1) (2) Pairs with one of
        // the (x-1) others. For every
        // pairing, number of people
        // reduce to (x-2).
        else
            return numberOfWays(x - 1) +
            (x - 1) * numberOfWays(x - 2);
    }

    // Driver code
    public static void Main ()
    {
        int x = 3;

        Console.WriteLine(numberOfWays(x));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Number of ways in which
// participant can take part.

function numberOfWays($x)
{
    // Base condition
    if ($x == 0 || $x == 1)
        return 1;

    // A participant can choose
    // to consider (1) Remains single.
    // Number of people reduce to (x-1)
    // (2) Pairs with one of the (x-1)
    // others. For every pairing, number
    // of peopl reduce to (x-2).
    else
        return numberOfWays($x - 1) +
            ($x - 1) * numberOfWays($x - 2);
}

// Driver code
$x = 3;
echo numberOfWays($x);

// This code is contributed by Sam007
?>
```

Output :

```
4
```

Since there are overlapping subproblems, we can optimize it usingdynamic programming.

C++

```
 // Number of ways in which participant can take part.
#include<iostream>
using namespace std;

int numberOfWays(int x)
{
    int dp[x+1];
    dp[0] = dp[1] = 1;

    for (int i=2; i<=x; i++)
       dp[i] = dp[i-1] + (i-1)*dp[i-2];

    return dp[x];
}

// Driver code
int main()
{
    int x = 3;
    cout << numberOfWays(x) << endl;
    return 0;
}
```

Java

```
 // Number of ways in which
// participant can take part.
import java.io.*;
class GFG {

static int numberOfWays(int x)
{
    int dp[] = new int[x+1];
    dp[0] = dp[1] = 1;

    for (int i=2; i<=x; i++)
    dp[i] = dp[i-1] + (i-1)*dp[i-2];

    return dp[x];
}

// Driver code
public static void main (String[] args) {
int x = 3;
System.out.println(numberOfWays(x));

    }
}
// This code is contributed by vt_m.
```

Python3

```
 # Python program to find Number of ways
# in which participant can take part.

# Function to calculate number of ways.
def numberOfWays (x):

    # Base condition
    if x == 0 or x == 1:
        return 1

    # A participant can choose to consider
    # (1) Remains single. Number of people
    # reduce to (x-1)
    # (2) Pairs with one of the (x-1) others.
    # For every pairing, number of people
    # reduce to (x-2).
    else:
        return (numberOfWays(x-1) +
              (x-1) * numberOfWays(x-2))

# Driver code
x = 3
print (numberOfWays(x))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
 // Number of ways in which
// participant can take part.
using System;

class GFG {

    static int numberOfWays(int x)
    {
        int []dp = new int[x+1];
        dp[0] = dp[1] = 1;

        for (int i = 2; i <= x; i++)
            dp[i] = dp[i - 1] +
                 (i - 1) * dp[i - 2];

        return dp[x];
    }

    // Driver code
    public static void Main ()
    {
        int x = 3;

        Console.WriteLine(numberOfWays(x));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for Number of ways
// in which participant can take part.

function numberOfWays($x)
{
    $dp[0] = 1;
    $dp[1] = 1;
    for ($i = 2; $i <= $x; $i++)
    $dp[$i] = $dp[$i - 1] + ($i - 1) *
                         $dp[$i - 2];

    return $dp[$x];
}

    // Driver code
    $x = 3;
    echo numberOfWays($x) ;

// This code is contributed by Sam007
?>
```

Output:

```
4
```

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/counting-pairs-person-can-form-pair-one/

Chapter 38

Counting sets of 1s and 0s in a binary matrix

Counting sets of 1s and 0s in a binary matrix - GeeksforGeeks

Given a n × m binary matrix, count the number of sets where a set can be formed one or more same values in a row or column.
Examples:

```
Input: 1 0 1
       0 1 0
Output: 8
Explanation: There are six one-element sets
(three 1s and three 0s). There are two two-
element sets, the first one consists of the
first and the third cells of the first row.
The second one consists of the first and the
third cells of the second row.

Input: 1 0
       1 1
Output: 6
```

The number of non-empty subsets of x elements is $2^x - 1$. We traverse every row and calculate numbers of 1's and 0's cells. For every u zeros and v ones, total sets is $2^u - 1 + 2^v - 1$. We then traverse all columns and compute same values and compute overall sum. We finally subtract m x n from the overall sum as single elements are considered twice.

CPP

```
 // CPP program to compute number of sets
```

```
// in a binary matrix.
#include <bits/stdc++.h>
using namespace std;

const int m = 3; // no of columns
const int n = 2; // no of rows

// function to calculate the number of
// non empty sets of cell
long long countSets(int a[n][m])
{
    // stores the final answer
    long long res = 0;

    // traverses row-wise
    for (int i = 0; i < n; i++)
    {
        int u = 0, v = 0;
        for (int j = 0; j < m; j++)
            a[i][j] ? u++ : v++;
        res += pow(2,u)-1 + pow(2,v)-1;
    }

    // traverses column wise
    for (int i = 0; i < m; i++)
    {
        int u = 0, v = 0;
        for (int j = 0; j < n; j++)
             a[j][i] ? u++ : v++;
        res += pow(2,u)-1 + pow(2,v)-1;
    }

    // at the end subtract n*m as no of
    // single sets have been added twice.
    return res-(n*m);
}

// driver program to test the above function.
int main() {

    int a[][3] = {(1, 0, 1),
                  (0, 1, 0)};

    cout << countSets(a);

    return 0;
}
```

Java

```
 // Java program to compute number of sets
// in a binary matrix.
class GFG {
static final int m = 3; // no of columns
static final int n = 2; // no of rows

// function to calculate the number of
// non empty sets of cell
static long countSets(int a[][]) {

    // stores the final answer
    long res = 0;

    // traverses row-wise
    for (int i = 0; i < n; i++) {
    int u = 0, v = 0;
    for (int j = 0; j < m; j++) {
        if (a[i][j] == 1)
        u++;
        else
        v++;
    }
    res += Math.pow(2, u) - 1 + Math.pow(2, v) - 1;
    }

    // traverses column wise
    for (int i = 0; i < m; i++) {
    int u = 0, v = 0;
    for (int j = 0; j < n; j++) {
        if (a[j][i] == 1)
        u++;
        else
        v++;
    }
    res += Math.pow(2, u) - 1 + Math.pow(2, v) - 1;
    }

    // at the end subtract n*m as no of
    // single sets have been added twice.
    return res - (n * m);
}

// Driver code
public static void main(String[] args) {
    int a[][] = {{1, 0, 1}, {0, 1, 0}};
    System.out.print(countSets(a));
}
}
// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to compute number of
// sets in a binary matrix.
using System;

class GFG {

    static int m = 3; // no of columns
    static int n = 2; // no of rows

    // function to calculate the number of
    // non empty sets of cell
    static long countSets(int [,]a)
    {

        // stores the final answer
        long res = 0;

        // Traverses row-wise
        for (int i = 0; i < n; i++)
        {
            int u = 0, v = 0;

            for (int j = 0; j < m; j++)
            {
                if (a[i,j] == 1)
                    u++;
                else
                    v++;
            }
            res += (long)(Math.Pow(2, u) - 1
                      + Math.Pow(2, v)) - 1;
        }

        // Traverses column wise
        for (int i = 0; i < m; i++)
        {
            int u = 0, v = 0;

            for (int j = 0; j < n; j++)
            {
                if (a[j,i] == 1)
```

```
                    u++;
                else
                    v++;
            }
            res += (long)(Math.Pow(2, u) - 1
                        + Math.Pow(2, v)) - 1;
        }

        // at the end subtract n*m as no of
        // single sets have been added twice.
        return res - (n * m);
    }

    // Driver code
    public static void Main()
    {
        int [,]a = {{1, 0, 1}, {0, 1, 0}};

        Console.WriteLine(countSets(a));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to compute
// number of sets
// in a binary matrix.

// no of columns
$m = 3;

// no of rows
$n = 2;

// function to calculate the number
// of non empty sets of cell
function countSets($a)
{
    global $m, $n;

    // stores the final answer
    $res = 0;

    // traverses row-wise
    for ($i = 0; $i < $n; $i++)
    {
        $u = 0; $v = 0;
        for ( $j = 0; $j < $m; $j++)
            $a[$i][$j] ? $u++ : $v++;
        $res += pow(2, $u) - 1 + pow(2, $v) - 1;
    }

    // traverses column wise
    for ($i = 0; $i < $m; $i++)
    {
        $u = 0;$v = 0;
        for ($j = 0; $j < $n; $j++)
            $a[$j][$i] ? $u++ : $v++;
        $res += pow(2, $u) - 1 +
                pow(2, $v) - 1;
    }

    // at the end subtract
    // n*m as no of single
    // sets have been added
    // twice.
    return $res-($n*$m);
}

    // Driver Code
    $a = array(array(1, 0, 1),
               array(0, 1, 0));

    echo countSets($a);

// This code is contributed by anuj_67.
?>
```

Output:

```
8
```

Time Complexity: O(n * m)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/counting-sets-of-1s-and-0s-in-a-binary-matrix/

Chapter 39

Cyclic Number

Cyclic Number - GeeksforGeeks

A cyclic number is an integer in which cyclic permutations of the digits are successive multiples of the number. The most widely known is the six-digit number 142857 (Please see below explanation given in examples).

The following trivial cases are typically excluded for Cyclic Numbers.

- Single digits, e.g.: 5
- Repeated digits, e.g.: 555
- Repeated cyclic numbers, e.g.: 142857142857

Given a number, check if it is cyclic or not.

Examples:

```
Input : 142857
Output : Yes
Explanation
    142857 × 1 = 142857
    142857 × 2 = 285714
    142857 × 3 = 428571
    142857 × 4 = 571428
    142857 × 5 = 714285
    142857 × 6 = 857142
```

We generate all cyclic permutations of the number and check if every permutation divides number of not. We also check for three conditions. If any of the three conditions is true, we return false.

C++

```
 // Program to check if a number is cyclic.
#include <bits/stdc++.h>
using namespace std;

#define ull unsigned long long int

// Function to generate all cyclic permutations
// of a number
bool isCyclic(ull N)
{
    // Count digits and check if all
    // digits are same
    ull num = N;
    int count = 0;
    int digit = num % 10;
    bool allSame = true;
    while (num) {
        count++;
        if (num % 10 != digit)
            allSame = false;
        num = num / 10;
    }

    // If all digits are same, then
    // not considered cyclic.
    if (allSame == true)
        return false;

    // If counts of digits is even and
    // two halves are same, then the
    // number is not considered cyclic.
    if (count % 2 == 0) {
        ull halfPower = pow(10, count / 2);
        ull firstHalf = N % halfPower;
        ull secondHalf = N / halfPower;
        if (firstHalf == firstHalf && isCyclic(firstHalf))
            return false;
    }

    num = N;
    while (1) {

        // Following three lines generates a
        // circular pirmutation of a number.
        ull rem = num % 10;
        ull div = num / 10;
        num = (pow(10, count - 1)) * rem + div;
        // If all the permutations are checked
        // and we obtain original number exit
        // from loop.
        if (num == N)
            break;

        if (num % N != 0)
            return false;
    }

    return true;
}

// Driver Program
int main()
{
    ull N = 142857;
    if (isCyclic(N))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java Program to check if a number is cyclic

class GFG {
    // Function to generate all cyclic
    // permutations of a number
    static boolean isCyclic(long N)
    {
        // Count digits and check if all
        // digits are same
        long num = N;
        int count = 0;
        int digit = (int)(num % 10);
        boolean allSame = true;
        while (num > 0) {
            count++;
            if (num % 10 != digit)
                allSame = false;
            num = num / 10;
        }

        // If all digits are same, then
        // not considered cyclic.
        if (allSame == true)
            return false;

        // If counts of digits is even and
        // two halves are same, then the
        // number is not considered cyclic.
        if (count % 2 == 0) {
            long halfPower = (long)Math.pow(10, count / 2);
            long firstHalf = N % halfPower;
            long secondHalf = N / halfPower;
            if (firstHalf == firstHalf && isCyclic(firstHalf))
                return false;
        }

        num = N;
        while (true) {
            // Following three lines generates a
            // circular pirmutation of a number.
            long rem = num % 10;
            long div = num / 10;
            num = (long)(Math.pow(10, count - 1))
                      * rem
                  + div;

            // If all the permutations are checked
            // and we obtain original number exit
            // from loop.
            if (num == N)
                break;

            if (num % N != 0)
                return false;
        }

        return true;
    }

    // Driver code
    public static void main(String[] args)
    {
        long N = 142857;
        if (isCyclic(N))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
 # Program to check if
# a number is cyclic
# Function to generate
# all cyclic permutations
# of a number
def isCyclic(N):

    # Count digits and check if all
    # digits are same
    num = N
    count = 0
    digit =(num % 10)
    allSame = True

    while (num>0):
        count+= 1
        if (num % 10 != digit):
            allSame = False
        num = num // 10


    # If all digits are same, then
    # not considered cyclic.
    if (allSame == True):
        return False

    # If counts of digits is even and
    # two halves are same, then the
    # number is not considered cyclic.
    if (count % 2 == 0):

        halfPower = pow(10, count//2)
        firstHalf = N % halfPower
        secondHalf = N / halfPower
        if (firstHalf == firstHalf and
            isCyclic(firstHalf)):
            return False


    num = N
    while (True):

        # Following three lines
        # generates a
        # circular pirmutation
        # of a number.
        rem = num % 10
        div = num // 10
        num = pow(10, count - 1) * rem + div

        # If all the permutations
        # are checked
        # and we obtain original
        # number exit
        # from loop.
        if (num == N):
            break

        if (num % N != 0):
            return False

    return True

# Driver code

N = 142857
if (isCyclic(N)):
    print("Yes")
else:
    print("No")

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Program to check if a number is cyclic
using System;

class GFG {

    // Function to generate all cyclic
    // permutations of a number
    static bool isCyclic(long N)
    {

        // Count digits and check if all
        // digits are same
        long num = N;
        int count = 0;
        int digit = (int)(num % 10);
        bool allSame = true;
```

```
while (num > 0)
{
    count++;
    if (num % 10 != digit)
        allSame = false;
    num = num / 10;
}

// If all digits are same, then
// not considered cyclic.
if (allSame == true)
    return false;

// If counts of digits is even and
// two halves are same, then the
// number is not considered cyclic.
if (count % 2 == 0) {
    long halfPower = (long)Math.Pow(10,
                        count / 2);

    long firstHalf = N % halfPower;

    // long secondHalf = N / halfPower;
    if (firstHalf == firstHalf &&
                   isCyclic(firstHalf))
        return false;
}

num = N;
while (true)
{

    // Following three lines generates a
    // circular pirmutation of a number.
    long rem = num % 10;
    long div = num / 10;
    num = (long)(Math.Pow(10, count - 1))
            * rem + div;

    // If all the permutations are checked
    // and we obtain original number exit
    // from loop.
    if (num == N)
        break;

    if (num % N != 0)
        return false;
}
```

```
        return true;
    }

    // Driver code
    public static void Main()
    {
        long N = 142857;

        if (isCyclic(N))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

Output:

```
Yes
```

Reference :
https://en.wikipedia.org/wiki/Cyclic_number

Improved By : vt_m

Source

https://www.geeksforgeeks.org/cyclic-number/

Chapter 40

Different ways to represent N as sum of K non-zero integers

Different ways to represent N as sum of K non-zero integers - GeeksforGeeks

Given N and K. The task is to find out how many different ways are there to represent N as the sum of K non-zero integers.

Examples:

> **Input:** N = 5, K = 3
> **Output:** 6
> The possible combinations of integers are:
> (1, 1, 3)
> (1, 3, 1)
> (3, 1, 1)
> (1, 2, 2)
> (2, 2, 1)
> (2, 1, 2)
>
> **Input:** N = 10, K = 4
> **Output:** 84

The **approach** to the problem is to observe a sequence and use combinations to solve the problem. To obtain a number N, N 1's are required, summation of N 1's will give N. The problem allows to use K integers only to make N.

Observation :

```
Let's take N = 5 and K = 3, then all
possible combinations of K numbers are: ( 1, 1, 3 )
                                        ( 1, 3, 1 )
                                        ( 3, 1, 1 )
                                        ( 1, 2, 2 )
```

```
                                  ( 2, 2, 1 )
                                  ( 2, 1, 2 )

The above can be rewritten as: ( 1, 1, 1 + 1 + 1 )
                               ( 1, 1 + 1 + 1, 1 )
                               ( 1 + 1 + 1, 1, 1 )
                               ( 1, 1 + 1, 1 + 1 )
                               ( 1 + 1, 1 + 1, 1 )
                               ( 1 + 1, 1, 1 + 1 )
```

From above, a conclusion can be drawn that of N 1's, k-1 commas have to be placed in between N 1's and the remaining places are to be filled with '+' signs. All combinations of placing k-1 commas and placing '+' signs in the remaining places will be the answer. So, in general, for N there will be N-1 spaces between all 1, and out of those choose k-1 and place a comma in between those 1. In between the rest 1's, place '+' signs. So ways of choosing K-1 objects out of N-1 is [illegible]. The dynamic programming approach is used to calculate [illegible].

Below is the implementation of the above approach:

C++

```
 // CPP program to calculate Different ways to
// represent N as sum of K non-zero integers.
#include <bits/stdc++.h>
using namespace std;

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int C[n + 1][k + 1];
    int i, j;

    // Caculate value of Binomial Coefficient in bottom up manner
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= min(i, k); j++) {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previosly stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }
    return C[n][k];
}

// Driver Code
int main()
{
    int n = 5, k = 3;
    cout << "Total number of different ways are "
         << binomialCoeff(n - 1, k - 1);
    return 0;
}
```

Java

```
 // Java program to calculate
// Different ways to represent
// N as sum of K non-zero integers.
import java.io.*;

class GFG
{

// Returns value of Binomial
// Coefficient C(n, k)
static int binomialCoeff(int n,
                         int k)
{
    int C[][] = new int [n + 1][k + 1];
    int i, j;

    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (j = 0;
             j <= Math.min(i, k); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using
            // previously stored values
            else
                C[i][j] = C[i - 1][j - 1] +
                          C[i - 1][j];
        }
    }

    return C[n][k];
}

// Driver Code
public static void main (String[] args)
{
    int n = 5, k = 3;
    System.out.println( "Total number of " +
                     "different ways are " +
                        binomialCoeff(n - 1,
                                      k - 1));
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# program to calculate
// Different ways to represent
// N as sum of K non-zero integers.
using System;

class GFG
{

// Returns value of Binomial
// Coefficient C(n, k)
static int binomialCoeff(int n,
                         int k)
{
    int [,]C = new int [n + 1,
                        k + 1];
    int i, j;

    // Calculate value of
    // Binomial Coefficient
    // in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (j = 0;
            j <= Math.Min(i, k); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
```

```
                C[i, j] = 1;

            // Calculate value using
            // previously stored values
            else
                C[i, j] = C[i - 1, j - 1] +
                          C[i - 1, j];
        }
    }

    return C[n,k];
}

// Driver Code
public static void Main ()
{
    int n = 5, k = 3;
    Console.WriteLine( "Total number of " +
                    "different ways are " +
                       binomialCoeff(n - 1,
                                  k - 1));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// PHP program to calculate
// Different ways to represent
// N as sum of K non-zero integers.

// Returns value of Binomial
// Coefficient C(n, k)
function binomialCoeff($n, $k)
{
    $C = array(array());
    $i; $j;

    // Caculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0;
             $j <= min($i, $k); $j++)
        {
            // Base Cases
            if ($j == 0 or $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using
            // previosly stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    return $C[$n][$k];
}

// Driver Code
$n = 5; $k = 3;
echo "Total number of " ,
  "different ways are " ,
   binomialCoeff($n - 1,
                 $k - 1);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
Total number of different ways are 6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/different-ways-to-represent-n-as-sum-of-k-non-zero-integers/

Chapter 41

Different ways to sum n using numbers greater than or equal to m

Different ways to sum n using numbers greater than or equal to m - GeeksforGeeks

Given two natural number **n** and **m**. The task is to find the number of ways in which the numbers that are greater than or equal to m can be added to get the sum n.

Examples:

```
Input : n = 3, m = 1
Output : 3
Following are three different ways
to get sum n such that each term is
greater than or equal to m
1 + 1 + 1, 1 + 2, 3

Input : n = 2, m = 1
Output : 2
Two ways are 1 + 1 and 2
```

The idea is to use Dynamic Programming by define 2D matrix, say dp[][]. **dp[i][j]** define the number of ways to get sum i using the numbers greater than or equal to j. So dp[i][j] can be defined as:

> **If i < j, dp[i][j] = 0**, because we cannot achieve smaller sum of i using numbers greater than or equal to j.
>
> **If i = j, dp[i][j] = 1**, because there is only one way to show sum i using number i which is equal to j.

> **Else dp[i][j] = dp[i][j+1] + dp[i-j][j]**, because obtaining a sum i using numbers greater than or equal to j is equal to the sum of obtaining a sum of i using numbers greater than or equal to j+1 and obtaining the sum of i-j using numbers greater than or equal to j.

Below is the implementation of this approach:

CPP

```
 // CPP Program to find number of ways to
// which numbers that are greater than
// given number can be added to get sum.
#include <bits/stdc++.h>
#define MAX 100
using namespace std;

// Return number of ways to which numbers
// that are greater than given number can
// be added to get sum.
int numberofways(int n, int m)
{
    int dp[n+2][n+2];
    memset(dp, 0, sizeof(dp));

    dp[0][n + 1] = 1;

    // Filling the table. k is for numbers
    // greater than or equal that are allowed.
    for (int k = n; k >= m; k--) {

        // i is for sum
        for (int i = 0; i <= n; i++) {

            // initializing dp[i][k] to number
            // ways to get sum using numbers
            // greater than or equal k+1
            dp[i][k] = dp[i][k + 1];

            // if i > k
            if (i - k >= 0)
                dp[i][k] = (dp[i][k] + dp[i - k][k]);
        }
    }

    return dp[n][m];
}
```

```
// Driver Program
int main()
{
    int n = 3, m = 1;
    cout << numberofways(n, m) << endl;
    return 0;
}
```

Java

```
 // Java Program to find number of ways to
// which numbers that are greater than
// given number can be added to get sum.
import java.io.*;

class GFG {

    // Return number of ways to which numbers
    // that are greater than given number can
    // be added to get sum.
    static int numberofways(int n, int m)
    {
        int dp[][]=new int[n+2][n+2];

        dp[0][n + 1] = 1;

        // Filling the table. k is for numbers
        // greater than or equal that are allowed.
        for (int k = n; k >= m; k--) {

            // i is for sum
            for (int i = 0; i <= n; i++) {

                // initializing dp[i][k] to number
                // ways to get sum using numbers
                // greater than or equal k+1
                dp[i][k] = dp[i][k + 1];

                // if i > k
                if (i - k >= 0)
                    dp[i][k] = (dp[i][k] + dp[i - k][k]);
            }
        }

        return dp[n][m];
    }

    // Driver Program
    public static void main(String args[])
    {
        int n = 3, m = 1;
        System.out.println(numberofways(n, m));
    }
}

/*This code is contributed by Nikita tiwari.*/
```

C#

```
 // C# program to find number of ways to
// which numbers that are greater than
// given number can be added to get sum.
using System;

class GFG {

    // Return number of ways to which numbers
    // that are greater than given number can
    // be added to get sum.
    static int numberofways(int n, int m)
    {
        int[, ] dp = new int[n + 2, n + 2];

        dp[0, n + 1] = 1;

        // Filling the table. k is for numbers
        // greater than or equal that are allowed.
        for (int k = n; k >= m; k--) {

            // i is for sum
            for (int i = 0; i <= n; i++) {

                // initializing dp[i][k] to number
                // ways to get sum using numbers
                // greater than or equal k+1
                dp[i, k] = dp[i, k + 1];

                // if i > k
                if (i - k >= 0)
                    dp[i, k] = (dp[i, k] + dp[i - k, k]);
            }
        }

        return dp[n, m];
    }
    // Driver Program
    public static void Main()
    {
        int n = 3, m = 1;
        Console.WriteLine(numberofways(n, m));
    }
}

/*This code is contributed by vt_m.*/
```

Output:

```
3
```

Source

https://www.geeksforgeeks.org/different-ways-sum-n-using-numbers-greater-equal-m/

Chapter 42

Dilworth's Theorem

Dilworth's Theorem - GeeksforGeeks

Let S be a finite partially ordered set. **The size of a maximal antichain equals the size of a minimal chain cover of S**. This is called the Dilworth's theorem. It is named after the mathematician Robert P. Dilworth(1950).

The width of a finite partially ordered set S is the maximum size of an antichain in S. In other words, the width of a finite partially ordered set S is the minimum number of chains needed to cover S, i.e. the minimum number of chains such that any element of S is in at least one of the chains.

Definition of chain : A chain in a partially ordered set is a subset of elements which are all comparable to each other.
Definition of antichain : An antichain is a subset of elements, no two of which are comparable to each other.

Illustrative examples :

```
Let S be the set of divisors of 30, with divisibility as the partial order.
Then the following chains cover S :
{1, 2, 6, 30}, {3, 15}, {5, 10}

And {2, 3, 5} is an antichain of length 3.
It is not immediately obvious, but the chain cover is minimal (though not unique),
and the antichain is maximal (though not unique).

So both definitions of width give 3 for this partially ordered set.
```

Proof of Dilworth's Theorem :
The easiest proof is by induction on the size of the set. Let d be the size of the largest antichain of S. The proof will show that S can be covered by d chains. The base case is trivial. So suppose the result has been proven for all sets smaller than S.

First, if no two elements of S are comparable, then S itself is an antichain and it can be covered by d = |S| chains each of length 1, so the result holds. Otherwise, let m be a minimal element (m <= z for all comparable z) and M be a maximal element (z <= M for all comparable z). Let T = S – {m, M}. If the largest antichain in T has size <= d – 1, then T can be covered by d – 1 chains, and so S can be covered by those plus the chain {m, M}, and the result will be proven for S.

Now, suppose that the largest antichain in T has size d(it can't be larger because T is a subset of S). Call this antichain A.

The idea of the rest of the proof is : picture the Hasse diagram for S where the largest antichain consists of a horizontal strip. Take everything below the strip and everything above the strip, use induction to cover these by chains, and then link the chains together by connecting them across the strip.

That is, construct the two sets

S^+ = {x belongs to S:x >= a for some a belongs to A}
S^- = {x belongs to S:x <= a for some a belongs to A}

Then S^+ U S^- must be all of S, because if it weren't then A would not be a maximal antichain in S. And S^+ U S^- = A, because if x is in the intersection, then a <= x <= b for some elements a, b belongs to A, so a and b are comparable by transitivity, so the only possibility is that a = b and they both equal x.

Since m and M are not in A, it must be the case that and m does not belong to S+, and m does not belong to S- so both sets), and the are strictly smaller than S. The inductive hypothesis applies to both S^- and S^+, so they are both covered by d chains, each of which must contain exactly one element of A. Call them C_a^- and C_a^+. Now we can stitch together these covers to get a cover of all of S, by the chains C_a^- U {a} U C_a^+. This cover has d chains, so the result follows by induction.

Code to illustrate the above example :
The classical O(N lg N) algorithm for longest increasing subsequence (LIS) can be seen as an application of Dilworth's Theorem. See here.

References : Youtube Video

Source

https://www.geeksforgeeks.org/dilworths-theorem/

Chapter 43

Distinct permutations of the string | Set 2

Distinct permutations of the string | Set 2 - GeeksforGeeks

Print all distinct permutation of a string having duplicates.

Examples:

```
Input : ABCA
Output : AABC AACB ABAC ABCA ACBA
         ACAB BAAC BACA BCAA CABA
         CAAB CBAA
```

An algorithm to print all distinct permutations has already been discussed here. Here we'll discuss one more approach to do the same. Recall first how me print permutations without any duplicates in the input string. It is given here. Let's now take the case of the string "ABAC". While generating permutations, let's say we are at index = 0, swap it with all elements after it. When we reach at i=2, we see that in the string s[index…i-1], there was an index which is equal to s[i]. Thus, swapping it will produce repeated permutations. Thus, we don't swap it. The below explains it better.

Illustration : Let us understand with below example.

```
i = 0 1 2 3
    A B A C
index = 0, s[0] = A
Start swapping s[index] with s[i] following it:
i = index + 1 = 1

Since s[index] != s[i], swap and recur.
```

```
i = 2, s[index] == s[i], don't swap

i = 3,  s[index] != s[i], swap and recur.
```

Below code does the same.

```
 #include <bits/stdc++.h>
using namespace std;

// Returns true if str[curr] does not matches with any of the
// characters after str[start]
bool shouldSwap(char str[], int start, int curr)
{
    for (int i = start; i < curr; i++)
        if (str[i] == str[curr])
            return 0;
    return 1;
}

// Prints all distinct permutations in str[0..n-1]
void findPermutations(char str[], int index, int n)
{
    if (index >= n) {
        cout << str << endl;
        return;
    }

    for (int i = index; i < n; i++) {

        // Proceed further for str[i] only if it
        // doesn't match with any of the characters
        // after str[index]
        bool check = shouldSwap(str, index, i);
        if (check) {
            swap(str[index], str[i]);
            findPermutations(str, index + 1, n);
            swap(str[index], str[i]);
        }
    }
}

// Driver code
int main()
{
    char str[] = "ABCA";
    int n = strlen(str);
    findPermutations(str, 0, n);
    return 0;
}
```

Output:

```
ABCA
ABAC
ACBA
ACAB
AACB
AABC
BACA
BAAC
BCAA
CBAA
CABA
CAAB
```

Improved By : sadiqRaza

Source

https://www.geeksforgeeks.org/distinct-permutations-string-set-2/

Chapter 44

Dividing an array into two halves of same sum

Dividing an array into two halves of same sum - GeeksforGeeks

Given an even size array of integers. We need to find if it is possible to divide array elements into two sets such that following conditions are true.

1. Size of both subsets is same.
2. Sum of elements in bot sets is same.
3. Every element is part of one of the two sets.

Examples :

```
Input: arr[] = {1, 3, 2, 1, 2, 1}
Output : Yes
Explanation: We can get two subsets
as {1, 3, 1} and {2, 2, 1}

Input: {1, 2, 3, 4, 5, 6}
Output : No
```

The idea is based on method 1 of following post.
Print all possible combinations of r elements in a given array of size n

We generate all subsets of size n/2. For every subset, we check if its sum is total sum by 2.

C++

```
 // Program to check if we can divide an array into two
```

```
// halves such that sum of the two is same.
#include <bits/stdc++.h>
using namespace std;

bool combinationUtil(int arr[], int half[], int start, int end,
                     int index, int n, int sum);

// Returns true if it is possible to divide array into two halves.
// of same sum. This function mainly uses combinationUtil()
bool isPossible(int arr[], int n)
{
    // If size of array is not even.
    if (n % 2 != 0)
        return false;

    // If sum of array is not even.
    int sum = accumulate(arr, arr + n, 0);
    if (sum % 2 != 0)
        return false;

    // A temporary array to store all combination one by one
    int half[n / 2];

    // Print all combination using temporary array 'half[]'
    return combinationUtil(arr, half, 0, n - 1, 0, n, sum);
}

/* arr[] ---> Input Array
   half[] ---> Temporary array to store current combination
               of size n/2
   start & end ---> Staring and Ending indexes in arr[]
   index ---> Current index in half[] */
bool combinationUtil(int arr[], int half[], int start, int end,
                     int index, int n, int sum)
{
    // Current combination is ready to be printed, print it
    if (index == n / 2) {
        int curr_sum = accumulate(half, half + n / 2, 0);
        return (curr_sum + curr_sum == sum);
    }

    // replace index with all possible elements. The condition
    // "end-i+1 >= n/2-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i = start; i <= end && end - i + 1 >= n/2 - index; i++) {
        half[index] = arr[i];
        if (combinationUtil(arr, half, i + 1, end, index + 1, n, sum))
```

```
            return true;
    }

    return false;
}

// Driver program to test above functions
int main()
{
    int arr[] = { 1, 2, 4, 4, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (isPossible(arr, n))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Java program to check if we can
// divide an array into two halves
// such that sum of the two is same.
public class DivideArray{

    static int accumulate(int arr[], int first,
                                     int last)
    {
        int init = 0;
        for (int i = first; i< last; i++) {
            init = init + arr[i];
        }

        return init;
    }

    // Returns true if it is possible to divide
    // array into two halves of same sum.
    // This function mainly uses combinationUtil()
    static Boolean isPossible(int arr[], int n)
    {
        // If size of array is not even.
        if (n % 2 != 0)
            return false;

        // If sum of array is not even.
        int sum = accumulate(arr, 0, n);
        if (sum % 2 != 0)
```

```
            return false;

        // A temporary array to store all
        // combination one by one int k=n/2;
        int half[] = new int[n/2];

        // Print all combination using temporary
        // array 'half[]'
        return combinationUtil(arr, half, 0, n - 1,
                                0, n, sum);
    }

    /* arr[] ---> Input Array
    half[] ---> Temporary array to store current
                combination of size n/2
    start & end ---> Staring and Ending indexes in arr[]
    index ---> Current index in half[] */
    static Boolean combinationUtil(int arr[], int half[],
                                   int start, int end,
                                   int index, int n,
                                   int sum)
    {
        // Current combination is ready to
        // be printed, print it
        if (index == n / 2) {
            int curr_sum = accumulate(half, 0 , n/2);
            return (curr_sum + curr_sum == sum);
        }

        // replace index with all possible elements.
        // The condition "end-i+1 >= n/2-index" makes
        // sure that including one element at index
        // will make a combination with remaining
        // elements at remaining positions
        for (int i = start; i <= end && end - i + 1 >=
                                 n/2 - index; i++) {
            half[index] = arr[i];
            if (combinationUtil(arr, half, i + 1, end,
                                 index + 1, n, sum))
                return true;
        }

        return false;
    }

    // Driver code
    public static void main(String[] s)
    {
        int arr[] = {1, 2, 4, 4, 5, 6 };

        if (isPossible(arr, arr.length))
        System.out.println("Yes");
        else
        System.out.println("NO");
    }
}

// This code is contributed by Prerna Saini
```

Output:

```
Yes
```

Source

https://www.geeksforgeeks.org/dividing-array-two-halves-sum/

Chapter 45

Entringer Number

Entringer Number - GeeksforGeeks

The **Entringer Number** E(n, k) are the number of permutations of {1, 2, …, n + 1}, starting with k + 1, which, after initally falling, alternatively fall then rise. The Entringer are given by:

$$E(n, k) = \begin{cases} 0 & \text{when } m>=n \text{ or } n=0 \\ 1 & m=0 \\ E(n, k-1) + E(n-1, k-n) & \text{otherwise} \end{cases}$$

For example, for n = 4 and k = 2, E(4, 2) is 4.
They are:
3 2 4 1 5
3 2 5 1 4
3 1 4 2 5
3 1 5 2 4

Examples :

```
Input : n = 4, k = 2
Output : 4

Input : n = 4, k = 3
Output : 5
```

Below is program to find Entringer Number E(n, k). The program is based on above simple recursive formula.

C++

```
 // CPP Program to find Entringer Number E(n, k)
#include <bits/stdc++.h>
using namespace std;

// Return Entringer Number E(n, k)
int zigzag(int n, int k)
{
    // Base Case
    if (n == 0 && k == 0)
        return 1;

    // Base Case
    if (k == 0)
        return 0;

    // Recursive step
    return zigzag(n, k - 1) +
           zigzag(n - 1, n - k);
}

// Driven Program
int main()
{
    int n = 4, k = 3;
    cout << zigzag(n, k) << endl;
    return 0;
}
```

Java

```
 // JAVA Code For Entringer Number
import java.util.*;

class GFG {

    // Return Entringer Number E(n, k)
    static int zigzag(int n, int k)
    {
        // Base Case
        if (n == 0 && k == 0)
            return 1;

        // Base Case
        if (k == 0)
            return 0;

        // Recursive step
        return zigzag(n, k - 1) +
               zigzag(n - 1, n - k);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 4, k = 3;
        System.out.println(zigzag(n, k));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python Program to find Entringer Number E(n, k)

# Return Entringer Number E(n, k)
def zigzag(n, k):

    # Base Case
    if (n == 0 and k == 0):
        return 1

    # Base Case
    if (k == 0):
        return 0

    # Recursive step
    return zigzag(n, k - 1) + zigzag(n - 1, n - k);

# Driven Program
n = 4
k = 3
print(zigzag(n, k))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Code For Entringer Number
using System;

class GFG {
    // Return Entringer Number E(n, k)
    static int zigzag(int n, int k)
    {
        // Base Case
        if (n == 0 && k == 0)
            return 1;

        // Base Case
        if (k == 0)
            return 0;

        // Recursive step
        return zigzag(n, k - 1) +
               zigzag(n - 1, n - k);
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 4, k = 3;
        Console.WriteLine(zigzag(n, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// Entringer Number E(n, k)

// Return Entringer Number E(n, k)
function zigzag($n, $k)
{
    // Base Case
    if ($n == 0 and $k == 0)
        return 1;

    // Base Case
    if ($k == 0)
        return 0;

    // Recursive step
    return zigzag($n, $k - 1) +
        zigzag($n - 1,$n - $k);
}
```

```
// Driven Code
$n = 4; $k = 3;
echo zigzag($n, $k) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
5
```

Below is the implementation of finding Entringer Number using Dynamic Programming:

C++

```
 // CPP Program to find Entringer Number E(n, k)
#include <bits/stdc++.h>
using namespace std;

// Return Entringer Number E(n, k)
int zigzag(int n, int k)
{
    int dp[n + 1][k + 1];
    memset(dp, 0, sizeof(dp));

    // Base cases
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++)
        dp[i][0] = 0;

    // Finding dp[i][j]
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++)
            dp[i][j] = dp[i][j - 1] +
                       dp[i - 1][i - j];

    return dp[n][k];
}

// Driven Program
int main()
{
    int n = 4, k = 3;
    cout << zigzag(n, k) << endl;
    return 0;
}
```

Java

```
 // JAVA Code For Entringer Number
import java.util.*;

class GFG {

    // Return Entringer Number E(n, k)
    static int zigzag(int n, int k)
    {
        int dp[][] = new int[n + 1][k + 1];

        // Base cases
        dp[0][0] = 1;
        for (int i = 1; i <= n; i++)
            dp[i][0] = 0;

        // Finding dp[i][j]
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= Math.min(i, k);
                                         j++)
                dp[i][j] = dp[i][j - 1] +
                           dp[i - 1][i - j];
            }

        return dp[n][k];
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 4, k = 3;
        System.out.println(zigzag(n, k));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Pyhton3 Program to find Entringer
# Number E(n, k)

# Return Entringer Number E(n, k)
def zigzag(n, k):
    dp = [[0 for x in range(k+1)]
             for y in range(n+1)]
    # Base cases
    dp[0][0] = 1
    for i in range(1, n+1):
        dp[i][0] = 0

    # Finding dp[i][j]
    for i in range(1, n+1):
        for j in range(1, k+1):
            dp[i][j] = (dp[i][j - 1]
                 + dp[i - 1][i - j])

    return dp[n][k]

# Driven Program
n = 4
k = 3
print(zigzag(n, k))

# This code is contributed by
# Prasad Kshirsagar
```

C#

```
 // C# Code For Entringer Number
using System;

class GFG {

    // Return Entringer Number E(n, k)
    static int zigzag(int n, int k)
    {
        int[, ] dp = new int[n + 1, k + 1];

        // Base cases
        dp[0, 0] = 1;
        for (int i = 1; i <= n; i++)
            dp[i, 0] = 0;

        // Finding dp[i][j]
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= Math.Min(i, k);
                j++)
                dp[i, j] = dp[i, j - 1] + dp[i - 1, i - j];
        }

        return dp[n, k];
    }
    /* Driver program to test above function */
    public static void Main()
    {
        int n = 4, k = 3;
        Console.WriteLine(zigzag(n, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// Entringer Number E(n, k)

// Return Entringer Number E(n, k)
function zigzag($n, $k)
{
    $dp = array(array());


    // Base cases
    $dp[0][0] = 1;
    for ($i = 1; $i <= $n; $i++)
        $dp[$i][0] = 0;

    // Finding dp[i][j]
    for ($i = 1; $i <= $n; $i++)
    {
        for ($j = 1; $j <= $i; $j++)
            $dp[$i][$j] = $dp[$i][$j - 1] +
                          $dp[$i - 1][$i - $j];
    }
    return $dp[$n][$k];
}

// Driven Code
$n = 4; $k = 3;
echo zigzag($n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

```
5
```

Improved By : vt_m, Prasad_Kshirsagar

Source

https://www.geeksforgeeks.org/entringer-number/

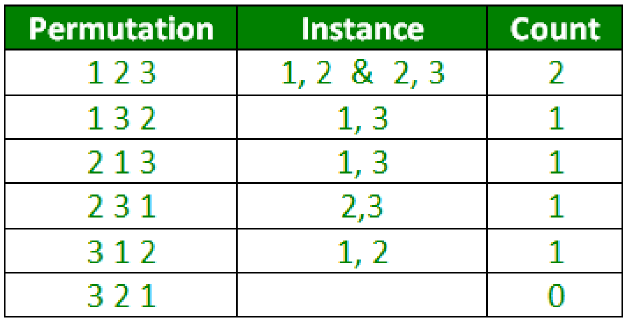
Chapter 46

Eulerian Number

Eulerian Number - GeeksforGeeks

In combinatorics, the **Eulerian Number** A(n, m), is the number of permutations of the numbers 1 to n in which exactly m elements are greater than previous element.

For example, there are 4 permutations of the number 1 to 3 in which exactly 1 element is greater than the previous elements.



Permutation	Instance	Count
1 2 3	1, 2 & 2, 3	2
1 3 2	1, 3	1
2 1 3	1, 3	1
2 3 1	2,3	1
3 1 2	1, 2	1
3 2 1		0

Examples :

```
Input : n = 3, m = 1
Output : 4
Please see above diagram (There
are 4 permutations where 1 no. is
greater.

Input : n = 4, m = 1
Output : 11
```

Eulerian Numbers are the coefficients of the Eulerian polynomials described below.

[illegible]

The Eulerian polynomials are defined by the exponential generating function

[illegible]

The Eulerian polynomials can be computed by the recurrence

[illegible]

[illegible]

An explicit formula for A(n, m) is

[illegible]

We can calculate A(n, m) by recurrence relation:

$$A(n,m)=\begin{cases}0 & \text{when } m>=n \text{ or } n=0\\ 1 & m=0\\ (n-m)A(n-1,m-1)+(m+1)A(n-1,m) & \text{otherwise}\end{cases}$$

Example:
Suppose, n = 3 and m = 1.
Therefore,
A(3, 1)
= (3 – 1) * A(2, 0) + (1 + 1) * A(2, 1)
= 2 * A(2, 0) + 2 * A(2, 1)
= 2 * 1 + 2 * ((2 – 1) * A(1, 0) + (1 + 1) * A(1, 1))
= 2 + 2 * (1 * 1 + 2 * ((1 – 1) * A(0, 0) + (1 + 1) * A(0, 1))
= 2 + 2 * (1 + 2 * (0 * 1 + 2 * 0)
= 2 + 2 * (1 + 2 * 0)
= 2 + 2 * 1
= 2 + 2
= 4
We can verify this with example shown above.

Below is the implementation of finding A(n, m):

C++

```
// CPP Program to find Eulerian number A(n, m)
```

```
#include <bits/stdc++.h>
using namespace std;

// Return euleriannumber A(n, m)
int eulerian(int n, int m)
{
    if (m >= n || n == 0)
        return 0;

    if (m == 0)
        return 1;

    return (n - m) * eulerian(n - 1, m - 1) +
           (m + 1) * eulerian(n - 1, m);
}

// Driven Program
int main()
{
    int n = 3, m = 1;
    cout << eulerian(n, m) << endl;
    return 0;
}
```

Java

```
 // Java rogram to find Eulerian number A(n, m)
import java.util.*;

class Eulerian
{
    // Return eulerian number A(n, m)
    public static int eulerian(int n, int m)
    {
        if (m >= n || n == 0)
            return 0;

        if (m == 0)
            return 1;

        return (n - m) * eulerian(n - 1, m - 1) +
            (m + 1) * eulerian(n - 1, m);
    }

    // driver code
    public static void main(String[] args)
    {
        int n = 3, m = 1;
        System.out.print( eulerian(n, m) );
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 Program to find Eulerian number A(n, m)

# Return euleriannumber A(n, m)
def eulerian(n, m):
    if (m >= n or n == 0):
        return 0;

    if (m == 0):
        return 1;

    return ((n - m) * eulerian(n - 1, m - 1) +
            (m + 1) * eulerian(n - 1, m))

# Driver code
n = 3
m = 1
print( eulerian(n, m) )

# This code is contributed by rishabh_jain
```

C#

```
 // C# rogram to find Eulerian number A(n, m)
using System;

class Eulerian {

    // Return eulerian number A(n, m)
    public static int eulerian(int n, int m)
    {
        if (m >= n || n == 0)
            return 0;

        if (m == 0)
            return 1;

        return (n - m) * eulerian(n - 1, m - 1) +
                    (m + 1) * eulerian(n - 1, m);
    }
    // driver code
    public static void Main()
    {
        int n = 3, m = 1;
        Console.WriteLine(eulerian(n, m));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP Program to find
// Eulerian number A(n, m)

// Return euleriannumber A(n, m)
function eulerian($n, $m)
{
    if ($m >= $n || $n == 0)
        return 0;

    if ($m == 0)
        return 1;

    return ($n - $m) * eulerian($n - 1, $m - 1) +
                ($m + 1) * eulerian($n - 1, $m);
}

// Driven Code
$n = 3; $m = 1;
echo eulerian($n, $m);

// This code is contributed by anuj_67.
?>
```

Output :

```
4
```

Below is the implementation of finding A(n, m) using Dynamic Programming:

C++

```
 // CPP Program to find Eulerian number A(n, m)
#include <bits/stdc++.h>
using namespace std;

// Return euleriannumber A(n, m)
int eulerian(int n, int m)
{
    int dp[n + 1][m + 1];

    memset(dp, 0, sizeof(dp));

    // For each row from 1 to n
    for (int i = 1; i <= n; i++) {

        // For each column from 0 to m
        for (int j = 0; j <= m; j++) {

            // If i is greater than j
            if (i > j) {

                // If j is 0, then make that
                // state as 1.
                if (j == 0)
                    dp[i][j] = 1;

                // basic recurrence relation.
                else
                    dp[i][j] = ((i - j) *
                     dp[i - 1][j - 1]) +
                    ((j + 1) * dp[i - 1][j]);
            }
        }
    }

    return dp[n][m];
}

// Driven Program
int main()
{
    int n = 3, m = 1;
    cout << eulerian(n, m) << endl;
    return 0;
}
```

Java

```
 // Java rogram to find Eulerian number A(n, m)
```

```
import java.util.*;

class Eulerian
{
    // Return euleriannumber A(n, m)
    public static int eulerian(int n, int m)
    {
        int[][] dp = new int[n+1][m+1];

        // For each row from 1 to n
        for (int i = 1; i <= n; i++) {

            // For each column from 0 to m
            for (int j = 0; j <= m; j++) {

                // If i is greater than j
                if (i > j) {

                    // If j is 0, then make
                    // that state as 1.
                    if (j == 0)
                        dp[i][j] = 1;

                    // basic recurrence relation.
                    else
                        dp[i][j] = ((i - j) *
                            dp[i - 1][j - 1]) +
                        ((j + 1) * dp[i - 1][j]);
                }
            }
        }

        return dp[n][m];
    }

    // driver code
    public static void main(String[] args)
    {
        int n = 3, m = 1;
        System.out.print( eulerian(n, m) );
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 Program to find Eulerian
```

```
# number A(n, m)

# Return euleriannumber A(n, m)
def eulerian(n, m):
    dp = [[0 for x in range(m+1)]
             for y in range(n+1)]

    # For each row from 1 to n
    for i in range(1, n+1):

        # For each column from 0 to m
        for j in range(0, m+1):

            # If i is greater than j
            if (i > j):
                # If j is 0, then make that
                # state as 1.

                if (j == 0):
                    dp[i][j] = 1

                # basic recurrence relation.
                else :
                    dp[i][j] = (((i - j) *
                       dp[i - 1][j - 1]) +
                       ((j + 1) * dp[i - 1][j]))

    return dp[n][m]

# Driven Program
n = 3
m = 1
print(eulerian(n, m))

# This code is contributed by Prasad Kshirsagar
```

C#

```
 // C# rogram to find Eulerian number A(n, m)
using System;

class Eulerian {

    // Return euleriannumber A(n, m)
    public static int eulerian(int n, int m)
    {
        int[, ] dp = new int[n + 1, m + 1];
        // For each row from 1 to n
        for (int i = 1; i <= n; i++) {

            // For each column from 0 to m
            for (int j = 0; j <= m; j++) {

                // If i is greater than j
                if (i > j) {

                    // If j is 0, then make
                    // that state as 1.
                    if (j == 0)
                        dp[i, j] = 1;

                    // basic recurrence relation.
                    else
                        dp[i, j] = ((i - j) * dp[i - 1, j - 1]) +
                                        ((j + 1) * dp[i - 1, j]);
                }
            }
        }

        return dp[n, m];
    }

    // driver code
    public static void Main()
    {
        int n = 3, m = 1;
        Console.WriteLine(eulerian(n, m));
    }
}

// This code is contributed by vt_m
```

Output :

```
4
```

Improved By : vt_m, Prasad_Kshirsagar

Source

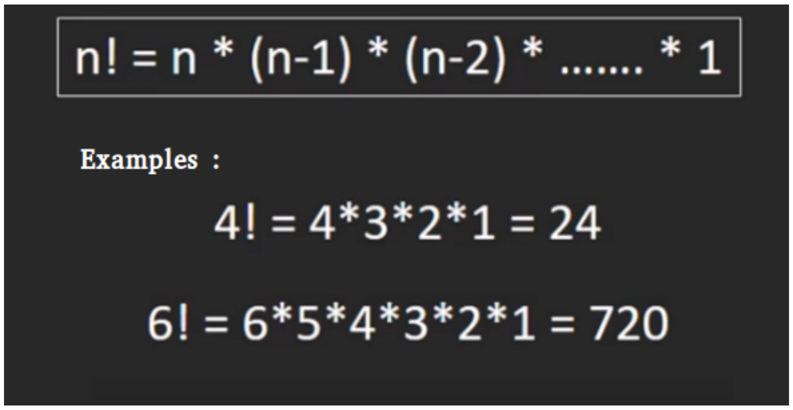
https://www.geeksforgeeks.org/eulerian-number/

Chapter 47

Factorial of a large number

Factorial of a large number - GeeksforGeeks

Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example factorial of 6 is 6*5*4*3*2*1 which is 720.



We have discussed simple program for factorial.

How to compute factorial of 100 using a C/C++ program?
Factorial of 100 has 158 digits. It is not possible to store these many digits even if we use long long int.

Examples :

```
Input : 100
Output : 933262154439441526816992388562667004-
         907159682643816214685929638952175999-
         932299156089414639761565182862536979-
         208272237582511852109168640000000000-
         00000000000000
```

```
Input :50
Output : 30414093201713378043612608166064768844-
         3776415689605120000000000000
```

Following is a simple solution where we use an array to store individual digits of the result. The idea is to use basic mathematics for multiplication.

The following is detailed algorithm for finding factorial.

factorial(n)
1) Create an array 'res[]' of MAX size where MAX is number of maximum digits in output.
2) Initialize value stored in 'res[]' as 1 and initialize 'res_size' (size of 'res[]') as 1.
3) Do following for all numbers from x = 2 to n.
……a) Multiply x with res[] and update res[] and res_size to store the multiplication result.

How to multiply a number 'x' with the number stored in res[]?
The idea is to use simple school mathematics. We one by one multiply x with every digit of res[]. The important point to note here is digits are multiplied from rightmost digit to leftmost digit. If we store digits in same order in res[], then it becomes difficult to update res[] without extra space. That is why res[] is maintained in reverse way, i.e., digits from right to left are stored.

multiply(res[], x)
1) Initialize carry as 0.
2) Do following for i = 0 to res_size – 1
….a) Find value of res[i] * x + carry. Let this value be prod.
….b) Update res[i] by storing last digit of prod in it.
….c) Update carry by storing remaining digits in carry.
3) Put all digits of carry in res[] and increase res_size by number of digits in carry.

```
Example to show working of multiply(res[], x)
A number 5189 is stored in res[] as following.
res[] = {9, 8, 1, 5}
x = 10

Initialize carry = 0;

i = 0, prod = res[0]*x + carry = 9*10 + 0 = 90.
res[0] = 0, carry = 9

i = 1, prod = res[1]*x + carry = 8*10 + 9 = 89
res[1] = 9, carry = 8

i = 2, prod = res[2]*x + carry = 1*10 + 8 = 18
res[2] = 8, carry = 1

i = 3, prod = res[3]*x + carry = 5*10 + 1 = 51
res[3] = 1, carry = 5
```

```
res[4] = carry = 5

res[] = {0, 9, 8, 1, 5}
```

Below is the implementation of above algorithm.

C++

```
 // C++ program to compute factorial of big numbers
#include<iostream>
using namespace std;

// Maximum number of digits in output
#define MAX 500

int multiply(int x, int res[], int res_size);

// This function finds factorial of large numbers
// and prints them
void factorial(int n)
{
    int res[MAX];

    // Initialize result
    res[0] = 1;
    int res_size = 1;

    // Apply simple factorial formula n! = 1 * 2 * 3 * 4...*n
    for (int x=2; x<=n; x++)
        res_size = multiply(x, res, res_size);

    cout << "Factorial of given number is \n";
    for (int i=res_size-1; i>=0; i--)
        cout << res[i];
}

// This function multiplies x with the number
// represented by res[].
// res_size is size of res[] or number of digits in the
// number represented by res[]. This function uses simple
// school mathematics for multiplication.
// This function may value of res_size and returns the
// new value of res_size
int multiply(int x, int res[], int res_size)
{
    int carry = 0;  // Initialize carry
    // One by one multiply n with individual digits of res[]
    for (int i=0; i<res_size; i++)
    {
        int prod = res[i] * x + carry;

        // Store last digit of 'prod' in res[]
        res[i] = prod % 10;

        // Put rest in carry
        carry  = prod/10;
    }

    // Put carry in res and increase result size
    while (carry)
    {
        res[res_size] = carry%10;
        carry = carry/10;
        res_size++;
    }
    return res_size;
}

// Driver program
int main()
{
    factorial(100);
    return 0;
}
```

Java

```
 // JAVA program to compute factorial
// of big numbers
class GFG {

    // This function finds factorial of
    // large numbers and prints them
    static void factorial(int n)
    {
        int res[] = new int[500];

        // Initialize result
        res[0] = 1;
        int res_size = 1;

        // Apply simple factorial formula
        // n! = 1 * 2 * 3 * 4...*n
        for (int x = 2; x <= n; x++)
            res_size = multiply(x, res, res_size);

        System.out.println("Factorial of given number is ");
        for (int i = res_size - 1; i >= 0; i--)
            System.out.print(res[i]);
    }

    // This function multiplies x with the number
    // represented by res[]. res_size is size of res[] or
    // number of digits in the number represented by res[].
    // This function uses simple school mathematics for
    // multiplication. This function may value of res_size
    // and returns the new value of res_size
    static int multiply(int x, int res[], int res_size)
    {
        int carry = 0; // Initialize carry

        // One by one multiply n with individual
        // digits of res[]
        for (int i = 0; i < res_size; i++)
        {
            int prod = res[i] * x + carry;
            res[i] = prod % 10; // Store last digit of
                                // 'prod' in res[]
            carry = prod/10; // Put rest in carry
        }

        // Put carry in res and increase result size
        while (carry!=0)
        {
            res[res_size] = carry % 10;
            carry = carry / 10;
            res_size++;
        }
        return res_size;
    }

    // Driver program
    public static void main(String args[])
    {
        factorial(100);
    }
}
//This code is contributed by Nikita Tiwari
```

Python

```
 # Python program to compute factorial
# of big numbers

import sys

# This function finds factorial of large
# numbers and prints them
def factorial( n) :
    res = [None]*500
    # Initialize result
    res[0] = 1
    res_size = 1

    # Apply simple factorial formula
    # n! = 1 * 2 * 3 * 4...*n
    x = 2
    while x <= n :
        res_size = multiply(x, res, res_size)
        x = x + 1

    print ("Factorial of given number is")
    i = res_size-1
    while i >= 0 :
        sys.stdout.write(str(res[i]))
        sys.stdout.flush()
        i = i - 1


# This function multiplies x with the number
# represented by res[]. res_size is size of res[]
# or number of digits in the number represented
# by res[]. This function uses simple school
# mathematics for multiplication. This function
# may value of res_size and returns the new value
# of res_size
def multiply(x, res,res_size) :

    carry = 0 # Initialize carry

    # One by one multiply n with individual
    # digits of res[]
    i = 0
    while i < res_size :
        prod = res[i] *x + carry
        res[i] = prod % 10; # Store last digit of
                            # 'prod' in res[]
        carry = prod/10; # Put rest in carry
        i = i + 1
    # Put carry in res and increase result size
    while (carry) :
        res[res_size] = carry % 10
        carry = carry / 10
        res_size = res_size + 1

    return res_size

# Driver program
factorial(100)

#This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to compute
// factorial of big numbers
using System;

class GFG
{

    // This function finds factorial
    // of large numbers and prints them
    static void factorial(int n)
    {
        int []res = new int[500];

        // Initialize result
        res[0] = 1;
        int res_size = 1;

        // Apply simple factorial formula
        // n! = 1 * 2 * 3 * 4...*n
        for (int x = 2; x <= n; x++)
            res_size = multiply(x, res,
                                res_size);

        Console.WriteLine("Factorial of " +
                          "given number is ");
        for (int i = res_size - 1; i >= 0; i--)
            Console.Write(res[i]);
    }

    // This function multiplies x
    // with the number represented
    // by res[]. res_size is size
    // of res[] or number of digits
    // in the number represented by
    // res[]. This function uses
    // simple school mathematics for
    // multiplication. This function
    // may value of res_size and
    // returns the new value of res_size
    static int multiply(int x, int []res,
                        int res_size)
    {
        int carry = 0; // Initialize carry

        // One by one multiply n with
        // individual digits of res[]
        for (int i = 0; i < res_size; i++)
        {
            int prod = res[i] * x + carry;
            res[i] = prod % 10; // Store last digit of
                                // 'prod' in res[]
            carry = prod / 10; // Put rest in carry
        }

        // Put carry in res and
        // increase result size
        while (carry != 0)
        {
            res[res_size] = carry % 10;
            carry = carry / 10;
            res_size++;
        }
        return res_size;
    }

    // Driver Code
    static public void Main ()
    {

        factorial(100);
    }
}

// This code is contributed by ajit
```

Output :

```
Factorial of given number is
9332621544394415268169923885626670049071596826438162146859296389
```

```
521759999322991560894146397615651828625369792082722375825118521 0
916864000000000000000000000000
```

The above approach can be optimized in many ways. We will soon be discussing optimized solution for same.

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

Improved By : jit_t

Source

https://www.geeksforgeeks.org/factorial-large-number/

Chapter 48

Find all distinct subsets of a given set

Find all distinct subsets of a given set - GeeksforGeeks

Given a set of positive integers, find all its subsets. The set can contain duplicate elements, so any repeated subset should be considered only once in the output.

Examples:

```
Input:  S = {1, 2, 2}
Output:  {}, {1}, {2}, {1, 2}, {2, 2}, {1, 2, 2}

Explanation:
The total subsets of given set are -
{}, {1}, {2}, {2}, {1, 2}, {1, 2}, {2, 2}, {1, 2, 2}
Here {2} and {1, 2} are repeated twice so they are considered
only once in the output
```

Prerequisite: Power Set

The idea is to use a bit-mask pattern to generate all the combinations as discussed in previous post. But previous post will print duplicate subsets if the elements are repeated in the given set. To handle duplicate elements, we construct a string out of given subset such that subsets having similar elements will result in same string. We maintain a list of such unique strings and finally we decode all such string to print its individual elements.

Below is its C++ implementation –

```
 // C++ program to find all subsets of given set. Any
// repeated subset is considered only once in the output
#include <bits/stdc++.h>
using namespace std;
```

```
// Utility function to split the string using a delim. Refer -
// http://stackoverflow.com/questions/236129/split-a-string-in-c
vector<string> split(const string &s, char delim)
{
    vector<string> elems;
    stringstream ss(s);
    string item;
    while (getline(ss, item, delim))
        elems.push_back(item);

    return elems;
}

// Function to find all subsets of given set. Any repeated
// subset is considered only once in the output
int printPowerSet(int arr[], int n)
{
    vector<string> list;

    /* Run counter i from 000..0 to 111..1*/
    for (int i = 0; i < (int) pow(2, n); i++)
    {
        string subset = "";

        // consider each element in the set
        for (int j = 0; j < n; j++)
        {
            // Check if jth bit in the i is set. If the bit
            // is set, we consider jth element from set
            if ((i & (1 << j)) != 0)
                subset += to_string(arr[j]) + "|";
        }

        // if subset is encountered for the first time
        // If we use set<string>, we can directly insert
        if (find(list.begin(), list.end(), subset) == list.end())
            list.push_back(subset);
    }

    // consider every subset
    for (string subset : list)
    {
        // split the subset and print its elements
        vector<string> arr = split(subset, '|');
        for (string str: arr)
            cout << str << " ";
        cout << endl;
    }
}

// Driver code
int main()
{
    int arr[] = { 10, 12, 12 };
    int n = sizeof(arr)/sizeof(arr[0]);

    printPowerSet(arr, n);

    return 0;
}
```

Output:

```

10
12
10 12
12 12
10 12 12
```

This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.GeeksforGeeks.org or mail your article to contribute@GeeksforGeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Source

https://www.geeksforgeeks.org/find-distinct-subsets-given-set/

Chapter 49

Find n-th lexicographically permutation of a string | Set 2

Find n-th lexicographically permutation of a string | Set 2 - GeeksforGeeks

Given a string of length m containing lowercase alphabets only. We need to find the n-th permutation of string lexicographically.

Examples:

```
Input : str[] = "abc", n = 3
Output : Result = "bac"
Explanation : All possible permutation
in sorted order: abc, acb, bac, bca,
cab, cba

Input : str[] = "aba", n = 2
Output : Result = "aba"
Explanation : All possible permutation
in sorted order: aab, aba, baa
```

We have discussed how to find lexicographically n-th permutation using STL. Time complexity of previous approach is O(n * n!) which is quite high.

Here we use some Mathematical concept for solving this problem.

The idea is based on below facts.

- The total number of permutation of a string formed by N characters (all distinct) is N!
- The Total number of permutation of a string formed by N characters (where the frequency of character C1 is M1, C2 is M2… and so the frequency of character Ck is Mk) is N!/(M1! * M2! *….Mk!).

- The total number of permutation of a string formed by N characters (all distinct) after fixing the first character is (N-1)!

We first count frequencies of all characters in an array freq[]. Now from the first smallest character present in the string (smallest index i such that freq[i] > 0), we compute the number of maximum permutation possible after setting that particular i-th character as the first character. If this sum value is more than given n, then we set that character as the first result output character, decrement freq[i], and continue same for remaining n-1 characters. On the other hand, if the count is less than the required n, iterate for the next character in the frequency table and update the count over and over again until we find a character that produces a count greater than the required n.
Time Complexity: O(n) i.e. order of string length

C++

```
 // C++ program to print n-th permutation
#include <bits/stdc++.h>
using namespace std;

#define ll long long int

const int MAX_CHAR = 26;
const int MAX_FACT = 20;
ll fact[MAX_FACT];

// utility for calculating factorials
void precomputeFactorials()
{
    fact[0] = 1;
    for (int i = 1; i < MAX_FACT; i++)
        fact[i] = fact[i - 1] * i;
}

// function for nth permutation
void nPermute(char str[], int n)
{
    precomputeFactorials();

    // length of given string
    int len = strlen(str);

    // Count frequencies of all
    // characters
    int freq[MAX_CHAR] = { 0 };
    for (int i = 0; i < len; i++)
        freq[str[i] - 'a']++;

    // out string for output string
```

```
char out[MAX_CHAR];

// iterate till sum equals n
int sum = 0;
int k = 0;

// We update both n and sum in this
// loop.
while (sum != n) {

    sum = 0;
    // check for characters present in freq[]
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] == 0)
            continue;

        // Remove character
        freq[i]--;

        // calculate sum after fixing
        // a particuar char
        int xsum = fact[len - 1 - k];
        for (int j = 0; j < MAX_CHAR; j++)
            xsum /= fact[freq[j]];
        sum += xsum;

        // if sum > n fix that char as
        // present char and update sum
        // and required nth after fixing
        // char at that position
        if (sum >= n) {
            out[k++] = i + 'a';
            n -= (sum - xsum);
            break;
        }

        // if sum < n, add character back
        if (sum < n)
            freq[i]++;
    }
}

// if sum == n means this char will provide its
// greatest permutation as nth permutation
for (int i=MAX_CHAR-1; k < len && i >= 0; i--)
    if (freq[i]) {
        out[k++] = i + 'a';
        freq[i++]--;
        }

    // append string termination
    // character and print result
    out[k] = '\0';
    cout << out;
}

// Driver program
int main()
{
    int n = 2;
    char str[] = "geeksquiz";

    nPermute(str, n);
    return 0;
}
```

Java

```
 // Java program to print n-th permutation
public class PermuteString
{
    final static int MAX_CHAR = 26;
    final static int MAX_FACT = 20;
    static long fact[] = new long[MAX_FACT];

    // utility for calculating factorial
    static void precomputeFactorirals()
    {
        fact[0] = 1;
        for (int i = 1; i < MAX_FACT; i++)
            fact[i] = fact[i - 1] * i;
    }

    // function for nth permutation
    static void nPermute(String str, int n)
    {
        precomputeFactorirals();

        // length of given string
        int len = str.length();

        // Count frequencies of all
        // characters
        int freq[] = new int[MAX_CHAR];
        for (int i = 0; i < len; i++)
            freq[str.charAt(i) - 'a']++;
```

```
// out string for output string
String out = "";

// iterate till sum equals n
int sum = 10;
int k = 0;

// We update both n and sum in this
// loop.
while (sum >= n) {

    // check for characters present in freq[]
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] == 0)
            continue;

        // Remove character
        freq[i]--;

        // calculate sum after fixing
        // a particular char
        sum = 0;
        int xsum = (int) fact[len - 1 - k];
        for (int j = 0; j < MAX_CHAR; j++)
            xsum /=  fact[freq[j]];
        sum += xsum;

        // if sum > n fix that char as
        // present char and update sum
        // and required nth after fixing
        // char at that position
        if (sum >= n) {
            out += (char)(i + 'a');
            k++;
            n -= (sum - xsum);
            break;
        }

        // if sum < n, add character back
        if (sum < n)
            freq[i]++;
    }
}

// if sum == n means this char will provide its
// greatest permutation as nth permutation
for (int i = MAX_CHAR - 1; k < len && i >= 0; i--)
```

```
            if (freq[i] != 0) {
                out += (char)(i + 'a');
                freq[i++]--;
            }

        // append string termination
        // character and print result
        System.out.println(out);
    }

    // Driver program to test above method
    public static void main(String[] args) {

        // TODO Auto-generated method stub
        int n = 2;
        String str = "geeksquiz";

        nPermute(str, n);
    }
}
// This code is contributed by Sumit Ghosh
```

C#

```
 // C# program to print n-th permutation
using System;

public class GFG {

    static int MAX_CHAR = 26;
    static int MAX_FACT = 20;
    static long []fact = new long[MAX_FACT];

    // utility for calculating factorial
    static void precomputeFactorirals()
    {
        fact[0] = 1;
        for (int i = 1; i < MAX_FACT; i++)
            fact[i] = fact[i - 1] * i;
    }

    // function for nth permutation
    static void nPermute(String str, int n)
    {
        precomputeFactorirals();

        // length of given string
        int len = str.Length;
        // Count frequencies of all
        // characters
        int []freq = new int[MAX_CHAR];

        for (int i = 0; i < len; i++)
            freq[str[i] - 'a']++;

        // out string for output string
        string ou = "";

        // iterate till sum equals n
        int sum = 10;
        int k = 0;

        // We update both n and sum in this
        // loop.
        while (sum >= n) {

            // check for characters present in freq[]
            for (int i = 0; i < MAX_CHAR; i++) {
                if (freq[i] == 0)
                    continue;

                // Remove character
                freq[i]--;

                // calculate sum after fixing
                // a particular char
                sum = 0;
                int xsum = (int) fact[len - 1 - k];

                for (int j = 0; j < MAX_CHAR; j++)
                    xsum /= (int)(fact[freq[j]]);

                sum += xsum;

                // if sum > n fix that char as
                // present char and update sum
                // and required nth after fixing
                // char at that position
                if (sum >= n) {
                    ou += (char)(i + 'a');
                    k++;
                    n -= (sum - xsum);
                    break;
                }
                // if sum < n, add character back
                if (sum < n)
                    freq[i]++;
            }
        }

        // if sum == n means this char will provide its
        // greatest permutation as nth permutation
        for (int i = MAX_CHAR - 1; k < len && i >= 0; i--)
            if (freq[i] != 0) {
                ou += (char)(i + 'a');
                freq[i++]--;
            }

        // append string termination
        // character and print result
    Console.Write(ou);
    }

    // Driver program to test above method
    public static void Main() {

        // TODO Auto-generated method stub
        int n = 2;
        String str = "geeksquiz";

        nPermute(str, n);
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
eegikqszu
```

Improved By : nitin mittal, pvsdileep

Source

https://www.geeksforgeeks.org/find-n-th-lexicographically-permutation-string-set-2/

Chapter 50

Find sum of even index binomial coefficients

Find sum of even index binomial coefficients - GeeksforGeeks

Given a positive integer **n**. The task is to find the sum of even indexed binomial coefficient. That is,
$^nC_0 + {}^nC_2 + {}^nC_4 + {}^nC_6 + {}^nC_8 +$ ……….

Examples :

```
Input : n = 4
Output : 8
4C0 + 4C2 + 4C4
= 1 + 6 + 1
= 8

Input : n = 6
Output : 32
```

Method 1: (Brute Force)
The idea is to find all the binomial coefficient and find only the sum of even indexed value.

CPP

```
 // CPP Program to find sum
// of even index term
#include <bits/stdc++.h>
using namespace std;

// Return the sum of
// even index term
```

```
int evenSum(int n)
{
    int C[n + 1][n + 1];
    int i, j;

    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= min(i, n); j++) {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using
            // previously stored values
            else
                C[i][j] = C[i - 1][j - 1]
                            + C[i - 1][j];
        }
    }

    // finding sum of even index term.
    int sum = 0;
    for (int i = 0; i <= n; i += 2)
        sum += C[n][i];

    return sum;
}

// Driver Program
int main()
{
    int n = 4;
    cout << evenSum(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to find sum
// of even index term
import java.io.*;
import java.math.*;

class GFG {

    // Return the sum of
    // even index term
    static int evenSum(int n)
    {
        int C[][] = new int [n + 1][n + 1];
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++)
        {
            for (j = 0; j <= Math.min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // else Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1]
                                + C[i - 1][j];
            }
        }

        // finding sum of even index term.
        int sum = 0;
        for (i = 0; i <= n; i += 2)
            sum += C[n][i];

        return sum;
    }

    // Driver Program
    public static void main(String args[])
    {
        int n = 4;
        System.out.println(evenSum(n));
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python

```
 # Python Program to find sum of even index term
import math

# Return the sum of even index term
def evenSum(n) :
    # Creates a list containing n+1 lists,
    # each of n+1 items, all set to 0
    C = [[0 for x in range(n + 1)] for y in range(n + 1)]

    # Calculate value of Binomial Coefficient
    # in bottom up manner
    for i in range(0, n + 1):
        for j in range(0, min(i, n + 1)):
            # Base Cases
            if j == 0 or j == i:
                C[i][j] = 1

            # Calculate value using previously
            # stored values
            else:
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

    # Finding sum of even index term
    sum = 0;
    for i in range(0, n + 1):
        if n % 2 == 0:
            sum = sum + C[n][i]

    return sum

# Driver method
n = 4
print evenSum(n)


# This code is contributed by 'Gitanjali'.
```

C#

```
 // C# Program to find sum
// of even index term
using System;

class GFG {

    // Return the sum of
    // even index term
    static int evenSum(int n)
    {
        int [,]C = new int [n + 1,n + 1];
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++)
        {
            for (j = 0; j <= Math.Min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // else Calculate value using
                // previously stored values
                else
                    C[i,j] = C[i - 1,j - 1]
                            + C[i - 1,j];
            }
        }

        // finding sum of even index term.
        int sum = 0;
        for (i = 0; i <= n; i += 2)
            sum += C[n,i];

        return sum;
    }

    // Driver Program
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(evenSum(n));
    }
}

/*This code is contributed by vt_m.*/
```

PHP

```
 <?php
// PHP Program to find sum
// of even index term

// Return the sum of
// even index term
function evenSum($n)
{
    $C = array(array());
    $i; $j;
    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $n); $j++)
        {
            // Base Cases
            if ($j == 0 or $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using
            // previously stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    // finding sum of even index term.
    $sum = 0;
    for ( $i = 0; $i <= $n; $i += 2)
        $sum += $C[$n][$i];

    return $sum;
}

// Driver Code
$n = 4;
echo evenSum($n) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
8
```

Time Complexity : O(n^2)

Method 2: (Using Formula)
Sum of even indexed binomial coeffient :
[illegible]

Proof :

```
We know,
(1 + x)n = nC0 + nC1 x + nC2 x2 + ..... + nCn xn

Now put x = -x, we get
(1 - x)n = nC0 - nC1 x + nC2 x2 + ..... + (-1)n nCn xn

Now, adding both the above equation, we get,
(1 + x)n + (1 - x)n = 2 * [nC0 + nC2 x2 + nC4 x4 + .......]

Put x = 1
(1 + 1)n + (1 - 1)n = 2 * [nC0 + nC2 + nC4 + .......]
2n/2 = nC0 + nC2 + nC4 + .......
2n-1 = nC0 + nC2 + nC4 + .......
```

Below is the implementation of this approach :

C++

```
 // CPP Program to find sum even indexed Binomial
// Coefficient.
#include <bits/stdc++.h>
using namespace std;

// Returns value of even indexed Binomial Coefficient
// Sum which is 2 raised to power n-1.
int evenbinomialCoeffSum(int n)
{
    return (1 << (n - 1));
}

/* Drier program to test above function*/
int main()
{
    int n = 4;
    printf("%d", evenbinomialCoeffSum(n));
    return 0;
}
```

Java

```
 // Java Program to find sum even indexed
// Binomial Coefficient.
import java.io.*;

class GFG {
// Returns value of even indexed Binomial Coefficient
// Sum which is 2 raised to power n-1.
static int evenbinomialCoeffSum(int n)
{
    return (1 << (n - 1));
}

// Driver Code
public static void main(String[] args)
{
int n = 4;
    System.out.println(evenbinomialCoeffSum(n));
}
    }

// This code is contributed by 'Gitanjali'.
```

Python

```
 # Python program to find sum even indexed
# Binomial Coefficient
import math

# Returns value of even indexed Binomial Coefficient
# Sum which is 2 raised to power n-1.
def evenbinomialCoeffSum( n):

    return (1 << (n - 1))

# Driver method
if __name__ == '__main__':
    n = 4
    print evenbinomialCoeffSum(n)

# This code is contributed by 'Gitanjali'.
```

C#

```
 // C# Program to find sum even indexed
// Binomial Coefficient.
using System;

class GFG
{
    // Returns value of even indexed
    // Binomial Coefficient Sum which
    // is 2 raised to power n-1.
    static int evenbinomialCoeffSum(int n)
    {
        return (1 << (n - 1));
    }

    // Driver Code
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(evenbinomialCoeffSum(n));
    }
}

// This code is contributed by 'Vt_m'.
```

PHP

```
 <?php
// PHP Program to find sum
// even indexed Binomial
// Coefficient.

// Returns value of even indexed
// Binomial Coefficient Sum which
// is 2 raised to power n-1.
function evenbinomialCoeffSum( $n)
{
    return (1 << ($n - 1));
}

    // Driver Code
    $n = 4;
    echo evenbinomialCoeffSum($n);

// This code is contributed by anuj_67.
?>
```

Output :

```
8
```

Time Complexity : O(1)

Sum of odd index binomial coefficient
Using the above result we can easily prove that the sum of odd index binomial coefficient is also 2^{n-1}.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-sum-even-index-binomial-coefficients/

Chapter 51

Generate all binary permutations such that there are more or equal 1's than 0's before every point in all permutations

Generate all binary permutations such that there are more or equal 1's than 0's before every point in all permutations - GeeksforGeeks

Generate all permutations of given length such that every permutation has more or equal 1's than 0's in all prefixes of the permutation.

Examples:

```
Input: len = 4
Output: 1111 1110 1101 1100 1011 1010
Note that a permutation like 0101 can not be in output because
there are more 0's from index 0 to 2 in this permutation.

Input: len = 3
Output: 111 110 101

Input: len = 2
Output: 11 10
```

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

Like permutation generation problems, recursion is the simplest approach to solve this. We start with an empty string, attach 1 to it and recur. While recurring, if we find more 1's at any point, we append a 0 and make one more recursive call.

```
 // C++ program to generate all permutations of 1's and 0's such that
// every permutation has more 1's than 0's at all indexes.
#include <iostream>
#include <cstring>
using namespace std;

// ones & zeroes --> counts of 1's and 0's in current string 'str'
// len ---> desired length of every permutation
void generate(int ones, int zeroes, string str, int len)
{
    // If length of current string becomes same as desired length
    if (len == str.length())
    {
        cout << str << "  ";
        return;
    }

    // Append a 1 and recur
    generate(ones+1, zeroes, str+"1", len);

    // If there are more 1's, append a 0 as well, and recur
    if (ones > zeroes)
        generate(ones, zeroes+1, str+"0", len);
}

// Driver program to test above function
int main()
{
    string str = "";
    generate(0, 0, str, 4);
    return 0;
}
```

Output:

```
1111  1110  1101  1100  1011  1010
```

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

Source

https://www.geeksforgeeks.org/generate-binary-permutations-1s-0s-every-point-permutations/

Chapter 52

Generate all cyclic permutations of a number

Generate all cyclic permutations of a number - GeeksforGeeks

Given a number N, our task is to generate all the possible cyclic permutations of the number.

A cyclic permutation shifts all the elements of a set by a fixed offset. For a set with elements a_0, a_1, …, a_n, a cyclic permutation of one place to the left would yield a_1, …, a_n, a_0, and a cyclic permutation of one place to the right would yield a_n, a_0, a_1, ….

Examples:

```
Input :  123
Output : 123
         312
         231

Input :  5674
Output : 5674
         4567
         7456
         6745
```

The idea is to generate next permutation of a number using below formula.

```
    rem = num % 10;
    div = num / 10;
    num = (pow(10, n - 1)) * rem + div;
```

While repeating above steps, if we come back to original number, we stop and return.

C++

```
 // Program to generate all cyclic permutations
// of number
#include <bits/stdc++.h>
using namespace std;

// Function to count the total number of digits
// in a number.
int countdigits(int N)
{
    int count = 0;
    while (N) {
        count++;
        N = N / 10;
    }
    return count;
}

// Function to generate all cyclic permutations
// of a number
void cyclic(int N)
{
    int num = N;
    int n = countdigits(N);

    while (1) {
        cout << num << endl;

        // Following three lines generates a
        // circular pirmutation of a number.
        int rem = num % 10;
        int div = num / 10;
        num = (pow(10, n - 1)) * rem + div;

        // If all the permutations are checked
        // and we obtain original number exit
        // from loop.
        if (num == N)
            break;
    }
}

// Driver Program
int main()
{
    int N = 5674;
    cyclic(N);
    return 0;
}
```

Java

```
 // Java Program to generate all
// cyclic permutations of number
class GFG
{

    // Function to count the total number
    // of digits in a number.
    static int countdigits(int N)
    {
        int count = 0;
        while (N>0) {
            count++;
            N = N / 10;
        }
        return count;
    }

    // Function to generate all cyclic
    // permutations of a number
    static void cyclic(int N)
    {
        int num = N;
        int n = countdigits(N);

        while (true) {
            System.out.println(num);

            // Following three lines generates a
            // circular pirmutation of a number.
            int rem = num % 10;
            int dev = num / 10;
            num = (int)((Math.pow(10, n - 1)) *
                                rem + dev);

            // If all the permutations are
            // checked and we obtain original
            // number exit from loop.
            if (num == N)
                break;
        }
    }
    // Driver Program
    public static void main (String[] args) {
    int N = 5674;
    cyclic(N);
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

Python3

```
 # Python3 Program to
# generate all cyclic
# permutations of number
import math

# Function to count the
# total number of digits
# in a number.
def countdigits(N):
    count = 0;
    while (N):
        count = count + 1;
        N = int(math.floor(N / 10));
    return count;

# Function to generate
# all cyclic permutations
# of a number
def cyclic(N):
    num = N;
    n = countdigits(N);
    while (1):
        print(int(num));

        # Following three lines
        # generates a circular
        # permutation of a number.
        rem = num % 10;
        div = math.floor(num / 10);
        num = ((math.pow(10, n - 1)) *
                           rem + div);

        # If all the permutations
        # are checked and we obtain
        # original number exit from loop.
        if (num == N):
            break;

# Driver Code
N = 5674;
cyclic(N);

# This code is contributed by mits
```

C#

```
 // C# Program to generate all
// cyclic permutations of number
using System;

class GFG
{
    // Function to count the total number
    // of digits in a number.
    static int countdigits(int N)
    {
        int count = 0;
        while (N > 0) {
            count++;
            N = N / 10;
        }
        return count;
    }

    // Function to generate all cyclic
    // permutations of a number
    static void cyclic(int N)
    {
        int num = N;
        int n = countdigits(N);

        while (true) {
            Console.WriteLine(num);

            // Following three lines generates a
            // circular permutation of a number.
            int rem = num % 10;
            int dev = num / 10;
            num = (int)((Math.Pow(10, n - 1)) *
                                    rem + dev);

            // If all the permutations are
            // checked and we obtain original
            // number exit from loop.
            if (num == N)
                break;
        }
    }

    // Driver Program
    public static void Main ()
    {
      int N = 5674;
      cyclic(N);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP Program to generate all
// cyclic permutations of number

// Function to count the total
// number of digits in a number.
function countdigits($N)
{
    $count = 0;
    while ($N)
    {
        $count++;
        $N = floor($N / 10);
    }
    return $count;
}

// Function to generate all
// cyclic permutations of a number
function cyclic($N)
{
    $num = $N;
    $n = countdigits($N);

    while (1)
    {
        echo ($num);
        echo "\n" ;

        // Following three lines generates a
        // circular pirmutation of a number.
        $rem = $num % 10;
        $div = floor($num / 10);
        $num = (pow(10, $n - 1)) * $rem + $div;

        // If all the permutations are checked
        // and we obtain original number exit
        // from loop.
        if ($num == $N)
            break;
    }
}

    // Driver Code
    $N = 5674;
    cyclic($N);

// This code is contributed by nitin mittal
?>
```

Output:

```
 5674
 4567
 7456
 6745
```

Improved By : nitin mittal, Mithun Kumar

Source

https://www.geeksforgeeks.org/generate-cyclic-permutations-number/

Chapter 53

Generate all passwords from given character set

Generate all passwords from given character set - GeeksforGeeks

Given a set of characters generate all possible passwords from them. This means we should generate all possible permutations of words using the given characters, with repititions and also upto a given length.

Examples:

```
Input : arr[] = {a, b},
          len = 2.
Output :
a b aa ab ba bb
```

The solution is to use recursion on the given character array. The idea is to pass all possible lengths and an empty string initially to a helper function. In the helper function we keep appending all the characters one by one to the current string and recur to fill the remaining string till the desired length is reached.

It can be better visualized using the below recursion tree:

```
        (a, b)
         /   \
        a     b
       / \   / \
      aa  ab ba bb
```

Following is the implementation of the above method.

C++

```
 // C++ program to generate all passwords for given characters
#include <bits/stdc++.h>
using namespace std;

// int cnt;

// Recursive helper function, adds/removes characters
// until len is reached
void generate(char* arr, int i, string s, int len)
{
    // base case
    if (i == 0) // when len has been reached
    {
        // print it out
        cout << s << "\n";
        // cnt++;
        return;
    }

    // iterate through the array
    for (int j = 0; j < len; j++) {

        // Create new string with next character
        // Call generate again until string has
        // reached its len
        string appended = s + arr[j];
        generate(arr, i - 1, appended, len);
    }

    return;
}

// function to generate all possible passwords
void crack(char* arr, int len)
{
    // call for all required lengths
    for (int i = 1; i <= len; i++) {
        generate(arr, i, "", len);
    }
}

// driver function
int main()
{
    char arr[] = { 'a', 'b', 'c' };
    int len = sizeof(arr) / sizeof(arr[0]);
    crack(arr, len);

    //cout << cnt << endl;
    return 0;
}
// This code is contributed by Satish Srinivas.
```

Python 3

```
 # Python3 program to
# generate all passwords
# for given characters

# Recursive helper function,
# adds/removes characters
# until len is reached
def generate(arr, i, s, len):

    # base case
    if (i == 0): # when len has
                 # been reached

        # print it out
        print(s)
        return

    # iterate through the array
    for j in range(0, len):

        # Create new string with
        # next character Call
        # generate again until
        # string has reached its len
        appended = s + arr[j]
        generate(arr, i - 1, appended, len)

    return

# function to generate
# all possible passwords
def crack(arr, len):

    # call for all required lengths
    for i in range(1 , len + 1):
        generate(arr, i, "", len)

# Driver Code
```

```
arr = ['a', 'b', 'c' ]
len = len(arr)
crack(arr, len)

# This code is contributed by Smita.
```

Output:

```
a
b
c
aa
ab
ac
ba
bb
bc
ca
cb
cc
aaa
aab
aac
aba
abb
abc
aca
acb
acc
baa
bab
bac
bba
bbb
bbc
bca
bcb
bcc
caa
cab
cac
cba
cbb
cbc
cca
ccb
ccc
```

If we want to see the count of the words, we can uncomment the lines having cnt variable in the code. We can observe that it comes out to be $n^1 + n^2 + .. + n^n$, where n = len. Thus the time complexity of the program is also $O(n^1 + n^n)$, hence exponential. We can also check for a particular password
while generating and break the loop and return if it is found. We can also include other symbols to be generated and if needed remove duplicates by preprocessing the input using a HashTable.

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/generate-passwords-given-character-set/

Chapter 54

Generate permutations with only adjacent swaps allowed

Generate permutations with only adjacent swaps allowed - GeeksforGeeks

Given a string on length N. You can swap only the adjacent elements and each element can be swapped atmost once. Find the no of permutations of the string that can be generated after performing the swaps as mentioned.

Examples:

```
Input : 12345
Output : 12345 12354 12435 13245 13254
         21345 21354 21435
```

Source: Goldman Sachs Interview

Consider any i-th character in the string. There are two possibilities for this character:
1.) Don't swap it, i.e. don't do anything with this character and move to the next character.
2.) Swap it. As it can be swapped with its adjacent,
……..a.) Swap it with the next character. Because each character can be swapped atmost once, we'll move to the position (i+2).
……. b.) Swap it with the previous character – we don't need to consider this case separately as i-th character is the next character of (i-1)th which is same as the case 2.a.

```
 // CPP program to generate permutations with only
// one swap allowed.
#include <cstring>
#include <iostream>
using namespace std;

void findPermutations(char str[], int index, int n)
```

```
{
    if (index >= n || (index + 1) >= n) {
        cout << str << endl;
        return;
    }

    // don't swap the current position
    findPermutations(str, index + 1, n);

    // Swap with the next character and
    // revert the changes. As explained
    // above, swapping with previous is
    // is not needed as it anyways happens
    // for next character.
    swap(str[index], str[index + 1]);
    findPermutations(str, index + 2, n);
    swap(str[index], str[index + 1]);
}

// Driver code
int main()
{
    char str[] = { "12345" };
    int n = strlen(str);
    findPermutations(str, 0, n);
    return 0;
}
```

Output:

```
12345
12354
12435
13245
13254
21345
21354
21435
```

Source

https://www.geeksforgeeks.org/generate-permutations-adjacent-swaps-allowed/

Chapter 55

Get the kth smallest number using the digits of the given number

Get the kth smallest number using the digits of the given number - GeeksforGeeks

Given a non-negative number **n** and a value **k**. Find the **kth** smallest number that can be formed using the digits of the given number **n**. It is guaranteed that the **kth** smallest number can be formed. Note that the number could be very large and may not even fit into long long int.

Examples:

```
Input : n = 1234, k = 2
Output : 1243

Input : n = 36012679802, k = 4
Output : 10022366897
```

The idea is to first sort digits and find the smallest number, then find k-th permutation starting from smallest number. To sort digits, we use an frequency counting technique as number of digits are small.

```
 // C++ implementation to get the kth smallest
// number using the digits of the given number
#include <bits/stdc++.h>
using namespace std;

// function to get the smallest digit in 'num'
// which is greater than 0
```

```
char getSmallDgtGreaterThanZero(string num, int n)
{
    // 's_dgt' to store the smallest digit
    // greater than 0
        char s_dgt = '9';

    for (int i=0; i<n; i++)
        if (num[i] < s_dgt && num[i] != '0')
            s_dgt = num[i];

    // required smallest digit in 'num'
    return s_dgt;
}

// function to get the kth smallest number
string kthSmallestNumber(string num, int k)
{
    // FIND SMALLEST POSSIBLE NUMBER BY SORTING
    // DIGITS

    // count frequency of each digit
    int freq[10];
    string final_num = "";

    memset(freq, 0, sizeof(freq));
    int n = num.size();

    // counting frequency of each digit
    for (int i = 0; i < n; i++)
        freq[num[i] - '0']++;

    // get the smallest digit greater than 0
    char s_dgt = getSmallDgtGreaterThanZero(num, n);

    // add 's_dgt' to 'final_num'
    final_num += s_dgt;

    // reduce frequency of 's_dgt' by 1 in 'freq'
    freq[s_dgt - '0']--;

    // add each digit according to its frequency
    // to 'final_num'
    for (int i=0; i<10; i++)
        for (int j=1; j<=freq[i]; j++)
            final_num += (char)(i+48);

    // FIND K-TH PERMUTATION OF SMALLEST NUMBER
    for (int i=1; i<k; i++)
```

```
      next_permutation(final_num.begin(), final_num.end());

    // required kth smallest number
    return final_num;
}

// Driver program to test above
int main()
{
    string num = "36012679802";
    int k = 4;
    cout << kthSmallestNumber(num, k);
    return 0;
}
```

Output:

```
10022366897
```

Source

https://www.geeksforgeeks.org/get-the-kth-smallest-number-using-the-digits-of-the-given-number/

Chapter 56

Heap's Algorithm for generating permutations

Heap's Algorithm for generating permutations - GeeksforGeeks

Heap's algorithm is used to generate all permutations of n objects. The idea is to generate each permutation from the previous permutation by choosing a pair of elements to interchange, without disturbing the other **n-2** elements.
Following is the illustration of generating all permutation of n given numbers.

Example:

```
Input: 1 2 3
Output: 1 2 3
    2 1 3
    3 1 2
    1 3 2
    2 3 1
    3 2 1
```

Algorithm:

1. The algorithm generates (n-1)! permutations of the first n-1 elements, adjoining the last element to each of these. This will generate all of the permutations that end with the last element.
2. If n is odd, swap the first and last element and if n is even, then swap the ith element (i is the counter starting from 0) and the last element and repeat the above algorithm till i is less than n.
3. In each iteration, the algorithm will produce all the permutations that end with the current last element.

Implementation:

C

```
// C++ program to print all permutations using
// Heap's algorithm
#include <bits/stdc++.h>
using namespace std;

//Prints the array
void printArr(int a[],int n)
{
    for (int i=0; i<n; i++)
        cout << a[i] << " ";
    printf("\n");
}

// Generating permutation using Heap Algorithm
void heapPermutation(int a[], int size, int n)
{
    // if size becomes 1 then prints the obtained
    // permutation
    if (size == 1)
    {
        printArr(a, n);
        return;
    }

    for (int i=0; i<size; i++)
    {
        heapPermutation(a,size-1,n);

        // if size is odd, swap first and last
        // element
        if (size%2==1)
            swap(a[0], a[size-1]);

        // If size is even, swap ith and last
        // element
        else
            swap(a[i], a[size-1]);
    }
}

// Driver code
int main()
{
    int a[] = {1, 2, 3};
    int n = sizeof a/sizeof a[0];
    heapPermutation(a, n, n);
    return 0;
}
```

Java

```
 // Java program to print all permutations using
// Heap's algorithm
class HeapAlgo
{
    //Prints the array
    void printArr(int a[], int n)
    {
        for (int i=0; i<n; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }

    //Generating permutation using Heap Algorithm
    void heapPermutation(int a[], int size, int n)
    {
        // if size becomes 1 then prints the obtained
        // permutation
        if (size == 1)
            printArr(a,n);

        for (int i=0; i<size; i++)
        {
            heapPermutation(a, size-1, n);

            // if size is odd, swap first and last
            // element
            if (size % 2 == 1)
            {
                int temp = a[0];
                a[0] = a[size-1];
                a[size-1] = temp;
            }

            // If size is even, swap ith and last
            // element
            else
            {
                int temp = a[i];
                a[i] = a[size-1];
                a[size-1] = temp;
            }
        }
    }

    // Driver code
    public static void main(String args[])
    {
        HeapAlgo obj = new HeapAlgo();
        int a[] = {1,2,3};
        obj.heapPermutation(a, a.length, a.length);
    }
}

// This code has been contributed by Amit Khandelwal.
```

Output:

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

References:

1. "https://en.wikipedia.org/wiki/Heap%27s_algorithm#cite_note-3

Source

https://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/

Chapter 57

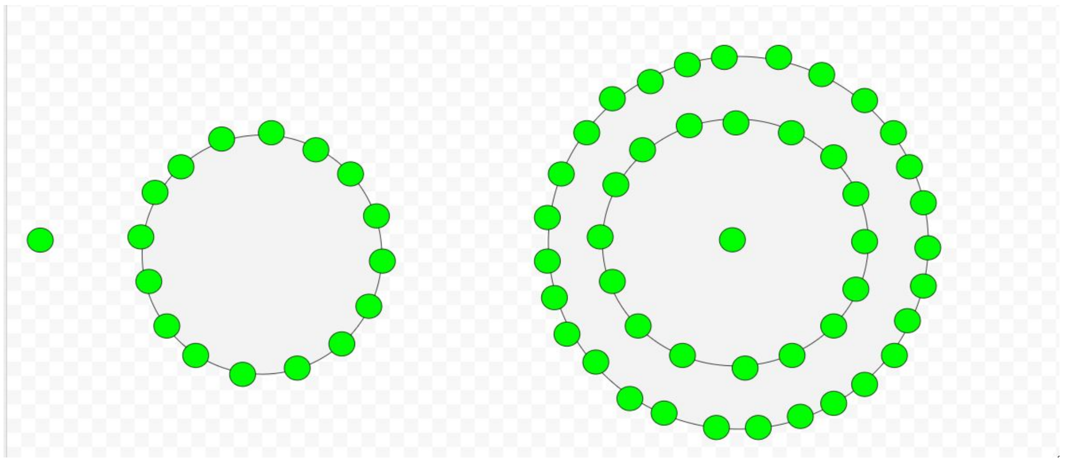
Hexadecagonal number

Hexadecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth hexadecagonal number.
A Hexadecagonal number is class of figurate number and a perfect squares. It has sixteen sided polygon called hexadecagon or hexakaidecagon. The n-th hexadecagonal number count's the sixteen number of dots and all others dots are surrounding to its successive layer.

Examples :

```
Input : 2
Output :16

Input :7
Output :301
```



Formula to calculate hexadecagonal number:

C++

```
 // C++ program to find Nth
// hexadecagon number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate hexadecagonal number
int hexadecagonalNum(long int n)
{
    return ((14 * n * n) - 12 * n) / 2;
}

// Drivers Code
int main()
{
    long int n = 5;
    cout << n << "th Hexadecagonal number : ";
    cout << hexadecagonalNum(n);
    cout << endl;
    n = 9;
    cout << n << "th Hexadecagonal number : ";
    cout << hexadecagonalNum(n);

    return 0;
}
```

Java

```
 // Java program to find Nth hexadecagon
// number
import java.io.*;

class GFG {

    // Function to calculate hexadecagonal
    // number
    static long hexadecagonalNum(long n)
    {
        return ((14 * n * n) - 12 * n) / 2;
    }

    // Drivers Code
    public static void main (String[] args)
    {
        long n = 5;
        System.out.println( n + "th "
          + "Hexadecagonal number : "
              + hexadecagonalNum(n));

        n = 9;
        System.out.println( n + "th "
          + "Hexadecagonal number : "
              + hexadecagonalNum(n));
    }
}

// This code contribued by anuj_67.
```

Python3

```
 # Python program to find Nth
# hexadecagon number

# Function to calculate
# hexadecagonal number
def hexadecagonalNum(n):

    # Formula to calculate nth
    # Centered heptagonal number
    return ((14 * n * n) - 12 * n) // 2

# Driver Code
n = 5
print("%sth Hexadecagonal number : " %n,
                    hexadecagonalNum(n))
n = 9
print("%sth Hexadecagonal number : " %n,
                    hexadecagonalNum(n))

# This code is contributed by ajit
```

C#

```
 // C# program to find Nth hexadecagon
// number
using System;
class GFG {

    // Function to calculate hexadecagonal
    // number
    static long hexadecagonalNum(long n)
    {
        return ((14 * n * n) - 12 * n) / 2;
    }

    // Drivers Code
    public static void Main ()
    {
        long n = 5;
        Console.WriteLine( n + "th "
        + "Hexadecagonal number : "
            + hexadecagonalNum(n));

        n = 9;
        Console.WriteLine( n + "th "
        + "Hexadecagonal number : "
            + hexadecagonalNum(n));
    }
}

// This code contribued by anuj_67.
```

PHP

```
 <?php
// PHP program to find Nth
// hexadecagon number

// Function to calculate
// hexadecagonal number

function hexadecagonalNum($n)
{
    return ((14 * $n * $n) - 12 * $n) / 2;
}

// Driver Code
$n = 5;
echo $n , "th Hexadecagonal number : ";
echo hexadecagonalNum($n);
echo "\n";

$n = 9;
echo $n , "th Hexadecagonal number : ";
echo hexadecagonalNum($n);

// This code is contributed bu m_kit
?>
```

Output :

```
5th Hexadecagonal number : 145
9th Hexadecagonal number : 513
```

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/hexadecagonal-number/

Chapter 58

How to find Lexicographically previous permutation?

How to find Lexicographically previous permutation? - GeeksforGeeks

Given a word, find lexicographically smaller permutation of it. For example, lexicographically smaller permutation of “4321” is “4312” and next smaller permutation of “4312” is “4231”. If the string is sorted in ascending order, the next lexicographically smaller permutation doesn’t exist.

We have discussed next_permutation() that modifies a string so that it stores lexicographically smaller permutation.
STL also provides std::prev_permutation. It returns ‘true’ if the function could rearrange the object as a lexicographically smaller permutation. Otherwise, it returns ‘false’.

```
 // C++ program to demonstrate working of
// prev_permutation()
#include <bits/stdc++.h>
using namespace std;

// Driver code
int main()
{
    string str = "4321";
    if ( prev_permutation(str.begin(), str.end()) )
        cout << "Previous permutation is "<< str ;
    else
        cout << "Previous permutation doesn't exist" ;
    return 0;
}
```

Output :

```
Previous permutation is 4312
```

How to write our own prev_permutation()?
Below are steps to find the previous permutation?
1. Find largest index i such that str[i – 1] > str[i].
2. Find largest index j such that j >= i and str[j] < str[i – 1].
3. Swap str[j] and str[i – 1].
4. Reverse the sub-array starting at str[i].

Below is C++ implementation of above steps.

```
 // C++ program to print all permutations with
// duplicates allowed using prev_permutation()
#include <bits/stdc++.h>
using namespace std;

// Function to compute the previous permutation
bool prevPermutation(string &str)
{
    // Find index of the last element of the string
    int n = str.length() - 1;

    // Find largest index i such that str[i ? 1] >
    // str[i]
    int i = n;
    while (i > 0 && str[i - 1] <= str[i])
        i--;

    // if string is sorted in ascending order
    // we're at the last permutation
    if (i <= 0)
        return false;

    // Note - str[i..n] is sorted in ascending order

    // Find rightmost element's index that is less
    // than str[i - 1]
    int j = i - 1;
    while (j + 1 <= n && str[j + 1] <= str[i - 1])
        j++;

    // Swap character at i-1 with j
    swap(str[i - 1], str[j]);

    // Reverse the substring [i..n]
    reverse(str.begin() + i, str.end());

    return true;
}

// Driver code
int main()
{
    string str = "4321";
    if ( prevPermutation(str) )
        cout << "Previous permutation is "<< str ;
    else
        cout << "Previous permutation doesn't exist" ;
    return 0;
}
```

Output :

```
Previous permutation is 4312
```

Source

https://www.geeksforgeeks.org/lexicographically-previous-permutation-in-c/

Chapter 59

Inclusion Exclusion principle and programming applications

Inclusion Exclusion principle and programming applications - GeeksforGeeks

Sum Rule – If a task can be done in one of n_1 ways or one of n_2 ways, where none of the set of n_1 ways is the same as any of the set of n_2 ways, then there are $n_1 + n_2$ ways to do the task.
The sum-rule mentioned above states that if there are multiple sets of ways of doing a task, there shouldn't be any way that is common between two sets of ways because if there is, it would be counted twice and the enumeration would be wrong.
The principle of **inclusion-exclusion** says that in order to count only unique ways of doing a task, we must add the number of ways to do it in one way and the number of ways to do it in another and then subtract the number of ways to do the task that are common to both sets of ways.
The principle of inclusion-exclusion is also known as the **subtraction principle**. For two sets of ways A_1 and A_2, the enumeration would like-

Below are some examples to explain the application of inclusion-exclusion principle:

- **Example 1:** How many binary strings of length 8 either start with a '1' bit or end with two bits '00'?
  Solution: If the string starts with one, there are 7 characters left which can be filled in $2^7 = 128$ ways.

  If the string ends with '00' then 6 characters can be filled in $2^6 = 64$ ways.
  Now if we add the above sets of ways and conclude that it is the final answer, then it would be wrong. This is because there are strings with start with '1' and end with

’00’ both, and since they satisfy both criteria they are counted twice.
So we need to subtract such strings to get a correct count.
Strings that start with ‘1’ and end with ’00’ have five characters that can be filled in $2^5 = 32$ ways.
So by the inclusion-exclusion principle we get-
Total strings = 128 + 64 – 32 = 160

- **Example 2:** How many numbers between 1 and 1000, including both, are divisible by 3 or 4?

  Solution: Number of numbers divisible by 3 = $|A_1| = \lfloor 1000/3 \rfloor = 333$.

  Number of numbers divisible by 4 = $|A_2| = \lfloor 1000/4 \rfloor = 250$.

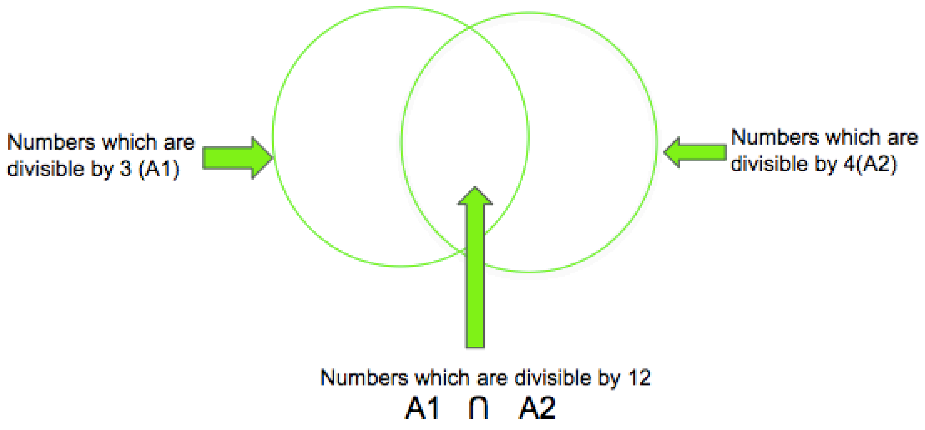
  Number of numbers divisible by 3 and 4 = $|A_1 \cap A_2| = \lfloor 1000/12 \rfloor = 83$.

  Therefore, number of numbers divisible by 3 or 4 = $|A_1 \cup A_2|$ = 333 + 250 – 83 = 500

Implementation

Problem 1: How many numbers between 1 and 1000, including both, are divisible by 3 or 4?

The Approach will be the one discussed above, we add the number of numbers that are divisible by 3 and 4 and subtract the numbers which are divisible by 12.


Numbers which are divisible by 3 (A1)
Numbers which are divisible by 4(A2)
Numbers which are divisible by 12
A1 ∩ A2

C++

```
// CPP program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
#include <bits/stdc++.h>
using namespace std;

// function to count the divisors
int countDivisors(int N, int a, int b)
{
    // Counts of numbers
    // divisible by a and b
    int count1 = N / a;
    int count2 = N / b;

    // inclusion-exclusion
    // principle applied
    int count3 = (N / (a * b));

    return count1 + count2 - count3;
}

// Driver Code
int main()
{
    int N = 1000, a = 3, b = 4;
    cout << countDivisors(N, a, b);
    return 0;
}
```

Java

```
 // Java program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
import java.io.*;

class GFG
{

// function to count the divisors
public static int countDivisors(int N,
                                int a,
                                int b)
{
    // Counts of numbers
    // divisible by a and b
    int count1 = N / a;
    int count2 = N / b;

    // inclusion-exclusion
    // principle applied
    int count3 = (N / (a * b));

    return count1 + count2 - count3;
}

// Driver Code
public static void main (String[] args)
{
    int N = 1000, a = 3, b = 4;
    System.out.println (countDivisors(N, a, b));
}
}

// This code is contributed by m_kit
```

C#

```
 // C# program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
using System;

class GFG
{

// function to count
// the divisors
public static int countDivisors(int N,
                                int a,
                                int b)
{
    // Counts of numbers
    // divisible by a and b
    int count1 = N / a;
    int count2 = N / b;

    // inclusion-exclusion
    // principle applied
    int count3 = (N / (a * b));

    return count1 + count2 - count3;
}
// Driver Code
static public void Main ()
{
    int N = 1000, a = 3, b = 4;
    Console.WriteLine(countDivisors(N, a, b));
}
}

// This code is contributed by aj_36
```

PHP

```
 <?php
// PHP program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4

// function to count the divisors
function countDivisors($N, $a, $b)
{
    // Counts of numbers
    // divisible by a and b
    $count1 = $N / $a;
    $count2 = $N / $b;

    // inclusion-exclusion
    // principle applied
    $count3 = ($N / ($a * $b));

    return $count1 + $count2 - $count3;
}

// Driver Code
$N = 1000; $a = 3; $b = 4;
echo countDivisors($N, $a, $b);

// This code is contributed by aj_36
?>
```

Output :

```
500
```

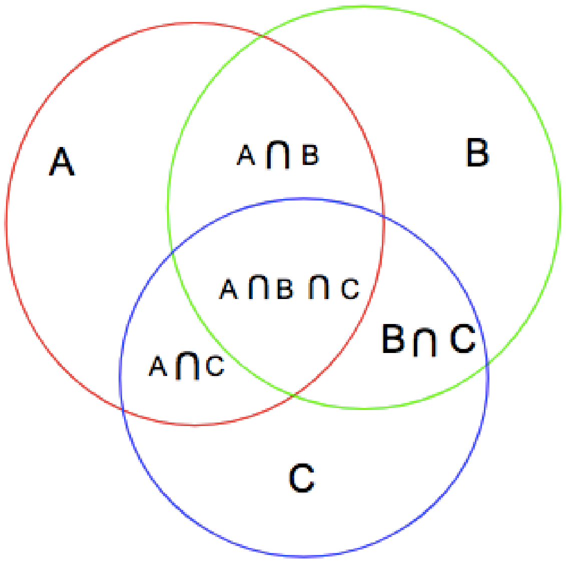
.

Problem 2: Given N prime numbers and a number M, find out how many numbers from 1 to M are divisible by any of the N given prime numbers.

Examples :

```
Input: N numbers = {2, 3, 5, 7}   M = 100
Output: 78

Input: N numbers = {2, 5, 7, 11}   M = 200
Output: 69
```

The approach for this problem will be to generate all the possible combinations of numbers using N prime numbers using power set in 2^N. For each of the given prime numbers P_i among N, **it has M/P_i multiples**. Suppose M=10, and we are given with 3 prime numbers(2, 3, 5), then the total count of multiples when we do 10/2 + 10/3 + 10/5 is 11. Since we are counting 6 and 10 twice, the count of multiples in range 1-M comes 11. Using **inclusion-exclusion principle**, we can get the correct number of multiples. The inclusion-exclusion principle for three terms can be described as:



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap$$

Similarly, for every N numbers, we can easily find the total number of multiples in range 1 to M by applying the formula for an intersection of N numbers. **The numbers that are formed by multiplication of an odd number of prime numbers will be added and**

the numbers formed by multiplication of even numbers will thus be subtracted to get the total number of multiples in the range 1 to M.

Using power set we can easily get all the combination of numbers formed by the given prime numbers. To know if the number is formed by multiplication of odd or even numbers, simply count the number of set bits in all the possible combinations **(1-1<<N)**.

Using power sets and adding the numbers created by combinations of odd and even prime numbers we get 123 and 45 respectively. Using **inclusion-exclusion principle** we get the number of numbers in range 1-M that is divided by any one of N prime numbers is **(odd combinations-even combinations) = (123-45) = 78.**
Below is the implementation of the above idea:

C++

```
 // CPP program to count the
// number of numbers in range
// 1-M that are divisible by
// given N prime numbers
#include <bits/stdc++.h>
using namespace std;

// function to count the number
// of numbers in range 1-M that
// are divisible by given N
// prime numbers
int count(int a[], int m, int n)
{
    int odd = 0, even = 0;
    int counter, i, j, p = 1;
    int pow_set_size = (1 << n);

    // Run from counter 000..0 to 111..1
    for (counter = 1; counter < pow_set_size;
                               counter++)
    {
        p = 1;
        for (j = 0; j < n; j++)
        {

            // Check if jth bit in the
            // counter is set If set
            // then pront jth element from set
            if (counter & (1 << j))
            {
                p *= a[j];
            }
        }
        // if set bits is odd, then add to
        // the number of multiples
        if (__builtin_popcount(counter) & 1)
            odd += (100 / p);
        else
            even += 100 / p;
    }

    return odd - even;
}
// Driver Code
int main()
{
    int a[] = { 2, 3, 5, 7 };
    int m = 100;
    int n = sizeof(a) / sizeof(a[0]);
    cout << count(a, m, n);
    return 0;
}
```

Output:

```
78
```

Time Complexity : O(2^N*N)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/inclusion-exclusion-principle-and-programming-applications/

Chapter 60

Inverse Permutation

Inverse Permutation - GeeksforGeeks

Given an array of size n of integers in range from 1 to n, we need to find the inverse permutation of that array.

An inverse permutation is a permutation which you will get by inserting position of an element at the position specified by the element value in the array. For better understanding, consider the following example:
Suppose we found element 4 at position 3 in an array, then in reverse permutation, we insert 3 (position of element 4 in the array) in position 4 (element value).
Basically, An inverse permutation is a permutation in which each number and the number of the place which it occupies is exchanged.

The array should contain element from 1 to array_size.

Example 1 :

```
Input  = {1, 4, 3, 2}
Output = {1, 4, 3, 2}
```

In this, For element 1 we insert position of 1 from arr1 i.e 1 at position 1 in arr2. For element 4 in arr1, we insert 2 from arr1 at position 4 in arr2. Similarly, for element 2 in arr1, we insert position of 2 i.e 4 in arr2.

```
Example 2 :
Input  = {2, 3, 4, 5, 1}
Output = {5, 1, 2, 3, 4}
```

In this example, for element 2 we insert position of 2 from arr1 in arr2 at position 2. similarly, we find the inverse permutation of other elements.

Consider an array arr having elements 1 to n.

Method 1 :
In this method, we take element one by one and check elements in increasing order and print the position of the element where we find that element.

C++

```
 // Naive CPP Program to find inverse permutation.
#include <bits/stdc++.h>
using namespace std;

// C++ function to find inverse permutations
void inversePermutation(int arr[], int size) {

  // Loop to select Elements one by one
  for (int i = 0; i < size; i++) {

    // Loop to print position of element
    // where we find an element
    for (int j = 0; j < size; j++) {

      // checking the element in increasing order
      if (arr[j] == i + 1) {

        // print position of element where
        // element is in inverse permutation
        cout << j + 1 << " ";
        break;
      }
    }
  }
}

// Driver program to test above function
int main() {
  int arr[] = {2, 3, 4, 5, 1};
  int size = sizeof(arr) / sizeof(arr[0]);
  inversePermutation(arr, size);
  return 0;
}
```

Java

```
 // Naive java Program to find inverse permutation.
import java.io.*;

class GFG {
    // java function to find inverse permutations
    static void inversePermutation(int arr[], int size)
    {
        int i ,j;
        // Loop to select Elements one by one
        for ( i = 0; i < size; i++)
        {

            // Loop to print position of element
            // where we find an element
            for ( j = 0; j < size; j++)
            {

                // checking the element in
                // increasing order
                if (arr[j] == i + 1)
                {
                    // print position of element
                    // where element is in inverse
                    // permutation
                    System.out.print( j + 1 + " ");
                    break;
                }
            }
        }
    }

    // Driver program to test above function


    public static void main (String[] args)
    {
        int arr[] = {2, 3, 4, 5, 1};
        int size = arr.length;
        inversePermutation(arr, size);

    }
}

// This code is contributed by vt_m
```

Python3

```
 # Naive Python3 Program to
# find inverse permutation.

# Function to find inverse permutations
def inversePermutation(arr, size):
    # Loop to select Elements one by one
    for i in range(0, size):

        # Loop to print position of element
        # where we find an element
        for j in range(0, size):

        # checking the element in increasing order
            if (arr[j] == i + 1):

                # print position of element where
                # element is in inverse permutation
                print(j + 1, end = " ")
                break

# Driver Code
arr = [2, 3, 4, 5, 1]
size = len(arr)

inversePermutation(arr, size)

#This code is cotributed by Smitha Dinesh Semwal
```

C#

```
 // Naive C# Program to find inverse permutation.
using System;

class GFG {

    // java function to find inverse permutations
    static void inversePermutation(int []arr, int size)
    {
        int i ,j;
        // Loop to select Elements one by one
        for ( i = 0; i < size; i++)
        {

            // Loop to print position of element
            // where we find an element
            for ( j = 0; j < size; j++)
            {

                // checking the element in
                // increasing order
                if (arr[j] == i + 1)
                {
                    // print position of element
                    // where element is in inverse
                    // permutation
                    Console.Write( j + 1 + " ");
                    break;
                }
            }
        }
    }

    // Driver program to test above function


    public static void Main ()
    {
        int []arr = {2, 3, 4, 5, 1};
        int size = arr.Length;
        inversePermutation(arr, size);

    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// Naive PHP Program to
// find inverse permutation.

// Function to find
// inverse permutations
function inversePermutation($arr, $size)
{
for ( $i = 0; $i < $size; $i++)
{

    // Loop to print position of element
    // where we find an element
    for ($j = 0; $j < $size; $j++)
    {

    // checking the element
    // in increasing order
    if ($arr[$j] == $i + 1)
    {

        // print position of element
        // where element is in
        // inverse permutation
        echo $j + 1 , " ";
        break;
    }
    }
}
}

// Driver Code
$arr = array(2, 3, 4, 5, 1);
$size = sizeof($arr);
inversePermutation($arr, $size);

// This code is contributed by aj_36
?>
```

Output :

```
 5 1 2 3 4
```

Method 2 :
The idea is to to use another array to store index and element mappings

C++

```
 // Efficient CPP Program to find inverse permutation.
#include <bits/stdc++.h>
using namespace std;

// C++ function to find inverse permutations
void inversePermutation(int arr[], int size) {

  // to store element to index mappings
  int arr2[size];

  // Inserting position at their
  // respective element in second array
  for (int i = 0; i < size; i++)
    arr2[arr[i] - 1] = i + 1;

  for (int i = 0; i < size; i++)
    cout << arr2[i] << " ";
}

// Driver program to test above function
int main() {
  int arr[] = {2, 3, 4, 5, 1};
  int size = sizeof(arr) / sizeof(arr[0]);
  inversePermutation(arr, size);
  return 0;
}
```

Java

```
 // Efficient Java Program to find
// inverse permutation.
import java.io.*;

class GFG {

// function to find inverse permutations
static void inversePermutation(int arr[], int size) {

    // to store element to index mappings
    int arr2[] = new int[size];

    // Inserting position at their
    // respective element in second array
    for (int i = 0; i < size; i++)
    arr2[arr[i] - 1] = i + 1;

    for (int i = 0; i < size; i++)
    System.out.print(arr2[i] + " ");
}

// Driver program to test above function
public static void main(String args[]) {
    int arr[] = {2, 3, 4, 5, 1};
    int size = arr.length;
    inversePermutation(arr, size);
}
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Efficient Python 3 Program to find
# inverse permutation.

# function to find inverse permutations
def inversePermutation(arr, size) :
    # To store element to index mappings
    arr2 = [0] *(size)

    # Inserting position at their
    # respective element in second array
    for i in range(0, size) :
        arr2[arr[i] - 1] = i + 1

    for i in range(0, size) :
        print( arr2[i], end = " ")

# Driver program
arr = [2, 3, 4, 5, 1]
size = len(arr)

inversePermutation(arr, size)

# This code is contributed by Nikita Tiwari.
```

C#

```
 // Efficient C# Program to find
// inverse permutation.
using System;

class GFG {

// function to find inverse permutations
static void inversePermutation(int []arr, int size) {

    // to store element to index mappings
    int []arr2 = new int[size];

    // Inserting position at their
    // respective element in second array
    for (int i = 0; i < size; i++)
    arr2[arr[i] - 1] = i + 1;

    for (int i = 0; i < size; i++)
    Console.Write(arr2[i] + " ");
}

// Driver program to test above function
public static void Main() {
    int []arr = {2, 3, 4, 5, 1};
    int size = arr.Length;
    inversePermutation(arr, size);
}
}

// This code is contributed by vt_m.
```

```
Output : 5 1 2 3 4
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/inverse-permutation/

Chapter 61

Johnson and Trotter algorithm

Johnson and Trotter algorithm - GeeksforGeeks

We are given a sequence of numbers from 1 to n. Each permutation in the sequence that we need to generate should differ from the previous permutation by swapping just two adjacent elements of the sequence.

Examples:

```
Input : n = 3
Output : 123 132 312 321 231 213

Input : n = 4
Output : 1234 1243 1423 4123 4132
1432 1342 1324 3124 3142 3412 4312
4321 3421 3241 3214 2314 2341 2431
4231 4213 2413 2143 2134
```

A **simple solution** to use permutations of n-1 elements to generate permutations of n elements.

For example let us see how to generate permutations of size 3 using permutations of size 2.

Permutations of two elements are 1 2 and 2 1.
Permutations of three elements can be obtained by inserting 3 at different positions in all permutations of size 2.
Inserting 3 in different positions of 1 2 leads to 1 2 **3**, 1 **3** 2 and **3** 1 2.
Inserting 3 in different positions of 2 1 leads to 2 1 **3**, 2 **3** 1 and **3** 2 1.

To generate permutations of size four, we consider all above six permutations of size three and insert 4 at different positions in every permutation.

Johnson and Trotter algorithm
The idea of Johnson and Trotter algorithm doesn't require to store all permutations of size n-1 and doesn't require going through all shorter permutations.

1. Find out the largest mobile integer in a particular sequence. **A directed integer is said to be mobile if it is greater than its immediate neighbor in the direction it is looking at**.
2. Switch this mobile integer and the adjacent integer to which its direction points.
3. Switch the direction of all the arrows whose value is greater than the mobile integer value.
4. Repeat the step 1 until unless there is no mobile integer left in the sequence.

```
Let us see how to generate permutations of
size four. We first print 1 2 3 4. Initially
all directions are right to left.
<1 <2 <3 <4

All mobile numbers are 2, 3 and 4. The largest
mobile number is 4. We swap 3 and 4. Since 4
is largest, we don't change any direction.
<1 <2 <4 <3

4 is the largest mobile. We swap it with 2.
<1 <4 <2 <3

4 is the largest mobile. We swap it with 1.
<4 <1 <2 <3

4 is the largest mobile. We swap it with 1.
<4 <1 <2 <3

3 is largest mobile. We swap it with 2 and
change directions of greater elements.
4> <1 <3 <2

4 is largest mobile. We swap it with 3.
<4 1> <2 <3

We proceed this way and print all permutations.
```

Below is the implementation of the approach.

C++

```
 // CPP program to print all permutations using
// Johnson and Trotter algorithm.
#include <bits/stdc++.h>
using namespace std;

bool LEFT_TO_RIGHT = true;
bool RIGHT_TO_LEFT = false;
```

```
// Utility functions for finding the
// position of largest mobile integer in a[].
int searchArr(int a[], int n, int mobile)
{
    for (int i = 0; i < n; i++)
        if (a[i] == mobile)
           return i + 1;
}

// To carry out step 1 of the algorithm i.e.
// to find the largest mobile integer.
int getMobile(int a[], bool dir[], int n)
{
    int mobile_prev = 0, mobile = 0;
    for (int i = 0; i < n; i++)
    {
        // direction 0 represents RIGHT TO LEFT.
        if (dir[a[i]-1] == RIGHT_TO_LEFT && i!=0)
        {
            if (a[i] > a[i-1] && a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }

        // direction 1 represents LEFT TO RIGHT.
        if (dir[a[i]-1] == LEFT_TO_RIGHT && i!=n-1)
        {
            if (a[i] > a[i+1] && a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    if (mobile == 0 && mobile_prev == 0)
        return 0;
    else
        return mobile;
}

// Prints a single permutation
int printOnePerm(int a[], bool dir[], int n)
{
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    // swapping the elements according to the
    // direction i.e. dir[].
    if (dir[a[pos - 1] - 1] ==  RIGHT_TO_LEFT)
       swap(a[pos-1], a[pos-2]);

    else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
       swap(a[pos], a[pos-1]);

    // changing the directions for elements
    // greater than largest mobile integer.
    for (int i = 0; i < n; i++)
    {
        if (a[i] > mobile)
        {
            if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                dir[a[i] - 1] = RIGHT_TO_LEFT;
            else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                dir[a[i] - 1] = LEFT_TO_RIGHT;
        }
    }

    for (int i = 0; i < n; i++)
        cout << a[i];
    cout << " ";
}

// To end the algorithm for efficiency it ends
// at the factorial of n because number of
// permutations possible is just n!.
int fact(int n)
{
    int res = 1;
    for (int i = 1; i <= n; i++)
        res = res * i;
    return res;
}

// This function mainly calls printOnePerm()
// one by one to print all permutations.
void printPermutation(int n)
{
    // To store current permutation
    int a[n];

    // To store current directions
    bool dir[n];
    // storing the elements from 1 to n and
    // printing first permutation.
    for (int i = 0; i < n; i++)
    {
        a[i] = i + 1;
        cout << a[i];
    }
    cout << endl;

    // initially all directions are set
    // to RIGHT TO LEFT i.e. 0.
    for (int i = 0; i < n; i++)
        dir[i] =  RIGHT_TO_LEFT;

    // for generating permutations in the order.
    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

// Driver code
int main()
{
    int n = 4;
    printPermutation(n);
    return 0;
}
```

Java

```
 // Java program to print all
// permutations using Johnson
// and Trotter algorithm.
import java.util.*;
import java.lang.*;

public class GfG{

    private final static boolean LEFT_TO_RIGHT = true;
    private final static boolean RIGHT_TO_LEFT = false;

    // Utility functions for
    // finding the position
    // of largest mobile
    // integer in a[].
    public static int searchArr(int a[], int n,
                                int mobile)
    {
```

```
    for (int i = 0; i < n; i++)

        if (a[i] == mobile)
            return i + 1;

    return 0;
}

// To carry out step 1
// of the algorithm i.e.
// to find the largest
// mobile integer.
public static int getMobile(int a[],
               boolean dir[], int n)
{
    int mobile_prev = 0, mobile = 0;

    for (int i = 0; i < n; i++)
    {
        // direction 0 represents
        // RIGHT TO LEFT.
        if (dir[a[i] - 1] == RIGHT_TO_LEFT &&
                                  i != 0)
        {
            if (a[i] > a[i - 1] &&
                        a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }

        // direction 1 represents
        // LEFT TO RIGHT.
        if (dir[a[i] - 1] == LEFT_TO_RIGHT &&
                              i ! =n - 1)
        {
            if (a[i] > a[i + 1] &&
                        a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    if (mobile == 0 && mobile_prev == 0)
        return 0;
    else
        return mobile;
}

// Prints a single
// permutation
public static int printOnePerm(int a[], boolean dir[],
                                                int n)
{
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    // swapping the elements
    // according to the
    // direction i.e. dir[].
    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
    {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    }
    else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
    {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

    // changing the directions
    // for elements greater
    // than largest mobile integer.
    for (int i = 0; i < n; i++)
    {
        if (a[i] > mobile)
        {
            if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                dir[a[i] - 1] = RIGHT_TO_LEFT;

            else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                dir[a[i] - 1] = LEFT_TO_RIGHT;
        }
    }

    for (int i = 0; i < n; i++)
        System.out.print(a[i]);

    System.out.print(" ");
    return 0;
}

// To end the algorithm
// for efficiency it ends
// at the factorial of n
// because number of
// permutations possible
// is just n!.
public static int fact(int n)
{
    int res = 1;

    for (int i = 1; i <= n; i++)
        res = res * i;
    return res;
}

// This function mainly
// calls printOnePerm()
// one by one to print
// all permutations.
public static void printPermutation(int n)
{
    // To store current
    // permutation
    int[] a = new int[n];

    // To store current
    // directions
    boolean[] dir = new boolean[n];

    // storing the elements
    // from 1 to n and
    // printing first permutation.
    for (int i = 0; i < n; i++)
    {
        a[i] = i + 1;
        System.out.print(a[i]);
    }

    System.out.print("\n");

    // initially all directions
    // are set to RIGHT TO
    // LEFT i.e. 0.
    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;
    // for generating permutations
    // in the order.
    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

// Driver function
public static void main(String argc[])
{
    int n = 4;
    printPermutation(n);
}

}

// This code is contributed by Sagar Shukla
```

C#

```
 // Java program to print all
// permutations using Johnson
// and Trotter algorithm.
using System;

public class GfG{

    private  static bool LEFT_TO_RIGHT = true;
    private  static bool RIGHT_TO_LEFT = false;

    // Utility functions for
    // finding the position
    // of largest mobile
    // integer in a[].
    public static int searchArr(int []a, int n,
                                int mobile)
    {
        for (int i = 0; i < n; i++)

            if (a[i] == mobile)
                return i + 1;

        return 0;
    }

    // To carry out step 1
    // of the algorithm i.e.
    // to find the largest
    // mobile integer.
    public static int getMobile(int []a,
                   bool []dir, int n)
    {
        int mobile_prev = 0, mobile = 0;

        for (int i = 0; i < n; i++)
        {
            // direction 0 represents
            // RIGHT TO LEFT.
            if (dir[a[i] - 1] == RIGHT_TO_LEFT &&
                                          i != 0)
            {
                if (a[i] > a[i - 1] &&
                            a[i] > mobile_prev)
                {
                    mobile = a[i];
                    mobile_prev = mobile;
                }
            }

            // direction 1 represents
            // LEFT TO RIGHT.
            if (dir[a[i] - 1] == LEFT_TO_RIGHT && i!=n - 1)
            {
                if (a[i] > a[i + 1] &&
                            a[i] > mobile_prev)
                {
                    mobile = a[i];
                    mobile_prev = mobile;
                }
            }
        }

        if (mobile == 0 && mobile_prev == 0)
            return 0;
        else
            return mobile;
    }

    // Prints a single
    // permutation
    public static int printOnePerm(int []a, bool []dir,
                                                     int n)
    {
        int mobile = getMobile(a, dir, n);
        int pos = searchArr(a, n, mobile);
```

```
        // swapping the elements
        // according to the
        // direction i.e. dir[].
        if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
        {
            int temp = a[pos - 1];
            a[pos - 1] = a[pos - 2];
            a[pos - 2] = temp;
        }
        else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
        {
            int temp = a[pos];
            a[pos] = a[pos - 1];
            a[pos - 1] = temp;
        }

        // changing the directions
        // for elements greater
        // than largest mobile integer.
        for (int i = 0; i < n; i++)
        {
            if (a[i] > mobile)
            {
                if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                    dir[a[i] - 1] = RIGHT_TO_LEFT;

                else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                    dir[a[i] - 1] = LEFT_TO_RIGHT;
            }
        }

        for (int i = 0; i < n; i++)
            Console.Write(a[i]);

        Console.Write(" ");

        return 0;
    }

    // To end the algorithm
    // for efficiency it ends
    // at the factorial of n
    // because number of
    // permutations possible
    // is just n!.
    public static int fact(int n)
    {
        int res = 1;
        for (int i = 1; i <= n; i++)
            res = res * i;
        return res;
    }

    // This function mainly
    // calls printOnePerm()
    // one by one to print
    // all permutations.
    public static void printPermutation(int n)
    {
        // To store current
        // permutation
        int []a = new int[n];

        // To store current
        // directions
        bool []dir = new bool[n];

        // storing the elements
        // from 1 to n and
        // printing first permutation.
        for (int i = 0; i < n; i++)
        {
            a[i] = i + 1;
            Console.Write(a[i]);
        }

        Console.Write("\n");

        // initially all directions
        // are set to RIGHT TO
        // LEFT i.e. 0.
        for (int i = 0; i < n; i++)
            dir[i] = RIGHT_TO_LEFT;

        // for generating permutations
        // in the order.
        for (int i = 1; i < fact(n); i++)
            printOnePerm(a, dir, n);
    }

    // Driver function
    public static void Main()
    {
        int n = 4;
        printPermutation(n);
    }

}

// This code is contributed by nitin mittal.
```

Output:

```
1234 1243 1423 4123 4132 1432 1342 1324
3124 3142 3412 4312 4321 3421 3241 3214
2314 2341 2431 4231 4213 2413 2143 2134
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/johnson-trotter-algorithm/

Chapter 62

K difference permutation

K difference permutation - GeeksforGeeks

Given two integers n and k. Consider first permutation of natural n numbers, P = "1 2 3 … n", print a permutation "Result" such that $\mathbf{abs(Result_i - P_i) = k}$ where P_i denotes the position of i in permutation P. The value of P_i varies from 1 to n. If there are multiple possible results, then print the lexicographically smallest one.

```
Input: n = 6 k = 3
Output: 4 5 6 1 2 3
Explanation:
     P = 1 2 3 4 5 6
Result = 4 5 6 1 2 3
We can notice that the difference between
individual numbers (at same positions) of
P and result is 3 and "4 5 6 1 2 3" is
lexicographically smallest such permutation.
Other greater permutations could be

Input  : n = 6 k = 2
Output : Not possible
Explanation: No permutation is possible
with difference is k
```

Naive approach is to generate all the permutation from 1 to n and pick the smallest one which satisfy the condition of absolute difference k. Time complexity of this approach is $\Omega(n!)$ which will definitely time out for large value of n.

The **Efficient approach** is to observe the pattern at each position of index. For each position of index i, there can only exist two candidate i.e., i + k and i − k. As we need to find lexicographically smallest permutation so we will first look for i − k candidate(if possible) and then for i + k candidate.

```
Illustration:
n = 8, k = 2
P : 1 2 3 4 5 6 7 8

For any ith position we will check which candidate
is possible i.e., i + k or i - k

1st pos = 1 + 2 = 3 (1 - 2 not possible)
2nd pos = 2 + 2 = 4 (2 - 2 not possible)
3rd pos = 3 - 2 = 1 (possible)
4th pos = 4 - 2 = 2 (possible)
5th pos = 5 + 2 = 7 (5 - 2 already placed, not possible)
6th pos = 6 + 2 = 8 (6 - 2 already placed, not possible)
7th pos = 7 - 2 = 5 (possible)
8th pos = 8 - 2 = 6 (possible)
```

Note: If we observe above illustration then we will find that i + k and i – k are alternating after k^{th} consecutive interval. Another observation is that the whole permutation is only when n is even such that n can be divided into two parts where each part must be divisible by k.

C++

```
 // C++ program to find k absolute difference
// permutation
#include<bits/stdc++.h>
using namespace std;

void kDifferencePermutation(int n, int k)
{
    // If k is 0 then we just print the
    // permutation from 1 to n
    if (!k)
    {
        for (int i = 0; i < n; ++i)
            cout << i + 1 << " ";
    }

    // Check whether permutation is feasible or not
    else if (n % (2 * k) != 0)
        cout <<"Not Possible";

    else
    {
        for (int i = 0; i < n; ++i)
        {
            // Put i + k + 1 candidate if position is
            // feasible, otherwise put the i - k - 1
            // candidate
            if ((i / k) % 2 == 0)
                cout << i + k + 1 << " ";
            else
                cout << i - k + 1 << " ";
        }
    }
    cout << "\n";
}

// Driver code
int main()
{
    int n = 6 , k = 3;
    kDifferencePermutation(n, k);

    n = 6 , k = 2;
    kDifferencePermutation(n, k);

    n = 8 , k = 2;
    kDifferencePermutation(n, k);

    return 0;
}
```

Java

```
 // Java program to find k absolute
// difference permutation
import java.io.*;

class GFG {

    static void kDifferencePermutation(int n,
                                int k)
    {
        // If k is 0 then we just print the
        // permutation from 1 to n
        if (!(k > 0))
        {
            for (int i = 0; i < n; ++i)
                System.out.print( i + 1 + " ");
        }

        // Check whether permutation is
        // feasible or not
        else if (n % (2 * k) != 0)
            System.out.print("Not Possible");

        else
        {
            for (int i = 0; i < n; ++i)
            {
                // Put i + k + 1 candidate
                // if position is feasible,
                // otherwise put the
                // i - k - 1 candidate
                if ((i / k) % 2 == 0)
                    System.out.print( i + k
                            + 1 + " ");
                else
                    System.out.print( i - k
                            + 1 + " ");
            }
        }
        System.out.println() ;
    }

    // Driver code
    static public void main (String[] args)
    {
        int n = 6 , k = 3;
        kDifferencePermutation(n, k);

        n = 6 ;
        k = 2;
        kDifferencePermutation(n, k);

        n = 8 ;
        k = 2;
        kDifferencePermutation(n, k);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to find k absolute
// difference permutation
using System;

class GFG {
```

```
static void kDifferencePermutation(int n,
                                   int k)
{
    // If k is 0 then we just print the
    // permutation from 1 to n
    if (!(k > 0))
    {
        for (int i = 0; i < n; ++i)
            Console.Write( i + 1 + " ");
    }

    // Check whether permutation is
    // feasible or not
    else if (n % (2 * k) != 0)
        Console.Write("Not Possible");

    else
    {
        for (int i = 0; i < n; ++i)
        {
            // Put i + k + 1 candidate
            // if position is feasible,
            // otherwise put the
            // i - k - 1 candidate
            if ((i / k) % 2 == 0)
                Console.Write( i + k
                          + 1 + " ");
            else
                Console.Write( i - k
                          + 1 + " ");
        }
    }
    Console.WriteLine() ;
}

// Driver code
static public void Main ()
{
    int n = 6 , k = 3;
    kDifferencePermutation(n, k);

    n = 6 ;
    k = 2;
    kDifferencePermutation(n, k);

    n = 8 ;
    k = 2;
    kDifferencePermutation(n, k);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find k absolute
// difference permutation

function kDifferencePermutation( $n, $k)
{

    // If k is 0 then we just print the
    // permutation from 1 to n
    if (!$k)
    {
        for($i = 0; $i < $n; ++$i)
            echo $i + 1 ," ";
    }

    // Check whether permutation
    // is feasible or not
    else if ($n % (2 * $k) != 0)
        echo"Not Possible";

    else
    {
        for($i = 0; $i < $n; ++$i)
        {

            // Put i + k + 1 candidate
            // if position is feasible,
            // otherwise put the i - k - 1
            // candidate
            if (($i / $k) % 2 == 0)
                echo $i + $k + 1 , " ";
            else
                echo $i - $k + 1 , " ";
        }
    }
    echo "\n";
}

    // Driver Code
    $n = 6 ; $k = 3;
    kDifferencePermutation($n, $k);
    $n = 6 ; $k = 2;
    kDifferencePermutation($n, $k);

    $n = 8 ;$k = 2;
    kDifferencePermutation($n, $k);

// This code is contributed by anuj_67.
?>
```

```
Output:
4 5 6 1 2 3
Not Possible
3 4 1 2 7 8 5 6
```

Time complexity: O(n)
Auxiliary space: O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/k-difference-permutation/

Chapter 63

Largest permutation after at most k swaps

Largest permutation after at most k swaps - GeeksforGeeks

Given a permutation of first **n** natural numbers as array and an integer k. Print the lexicographically largest permutation after at most k swaps

```
Input: arr[] = {4, 5, 2, 1, 3}
       k = 3
Output: 5 4 3 2 1
Explanation:
Swap 1st and 2nd elements: 5 4 2 1 3
Swap 3rd and 5th elements: 5 4 3 1 2
Swap 4th and 5th elements: 5 4 3 2 1

Input: arr[] = {2, 1, 3}
       k = 1
Output: 3 1 2
```

A **Naive approach** is to one by one generate permutation in lexicographically decreasing order. Compare every generated permutation with original array and count the number of swaps required to convert. If count is less than or equal to k, print this permutation. Problem of this approach is that it would be difficult to implement and will definitely time out for large value of N.

An **Efficient** approach is to think greedily. If we visualize the problem then we will get to know that largest permutation can only be obtained if it starts with n and continues with n-1, n-2,…. So we just need to put the 1st, 2nd, 3rd, …, kth largest element to their respective position.

C++

```
 // Below is C++ code to print largest permutation
// after atmost K swaps
#include<bits/stdc++.h>
using namespace std;

// Function to calculate largest permutation after
// atmost K swaps
void KswapPermutation(int arr[], int n, int k)
{
    // Auxiliary dictionary of storing the position
    // of elements
    int pos[n+1];

    for (int i = 0; i < n; ++i)
        pos[arr[i]] = i;

    for (int i=0; i<n && k; ++i)
    {
        // If element is already i'th largest,
        // then no need to swap
        if (arr[i] == n-i)
            continue;

        // Find position of i'th largest value, n-i
        int temp = pos[n-i];

        // Swap the elements position
        pos[arr[i]] = pos[n-i];
        pos[n-i] = i;

        // Swap the ith largest value with the
        // current value at ith place
        swap(arr[temp], arr[i]);

        // decrement number of swaps
        --k;
    }
}

// Driver code
int main()
{
    int arr[] = {4, 5, 2, 1, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;

    KswapPermutation(arr, n, k);
    cout << "Largest permutation after "
```

```
         << k << " swaps:n";
    for (int i=0; i<n; ++i)
        printf("%d ", arr[i]);
    return 0;
}
```

Java

```
 // Below is Java code to print largest permutation
// after atmost K swaps
class GFG {

    // Function to calculate largest permutation after
    // atmost K swaps
    static void KswapPermutation(int arr[], int n, int k)
    {

        // Auxiliary dictionary of storing the position
        // of elements
        int pos[] = new int[n+1];

        for (int i = 0; i < n; ++i)
            pos[arr[i]] = i;

        for (int i = 0; i < n && k > 0; ++i)
        {

            // If element is already i'th largest,
            // then no need to swap
            if (arr[i] == n-i)
                continue;

            // Find position of i'th largest value, n-i
            int temp = pos[n-i];

            // Swap the elements position
            pos[arr[i]] = pos[n-i];
            pos[n-i] = i;

            // Swap the ith largest value with the
            // current value at ith place
            int tmp1 = arr[temp];
            arr[temp] = arr[i];
            arr[i] = tmp1;

            // decrement number of swaps
            --k;
        }
    }

    // Driver method
    public static void main(String[] args)
    {

        int arr[] = {4, 5, 2, 1, 3};
        int n = arr.length;
        int k = 3;

        KswapPermutation(arr, n, k);

        System.out.print("Largest permutation "
                  + "after " + k + " swaps:\n");

        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
 # Python code to print largest permutation after K swaps

def KswapPermutation(arr, n, k):

    # Auxiliary array of storing the position of elements
    pos = {}
    for i in range(n):
        pos[arr[i]] = i

    for i in range(n):

        # If K is exhausted then break the loop
        if k == 0:
            break

        # If element is already largest then no need to swap
        if (arr[i] == n-i):
            continue

        # Find position of i'th largest value, n-i
        temp = pos[n-i]

        # Swap the elements position
        pos[arr[i]] = pos[n-i]
        pos[n-i] = i

        # Swap the ith largest value with the value at
        # ith place
        arr[temp], arr[i] = arr[i], arr[temp]

        # Decrement K after swap
        k = k-1

# Driver code
arr = [4, 5, 2, 1, 3]
n = len(arr)
k = 3
KswapPermutation(arr, N, K)

# Print the answer
print "Largest permutation after", K, "swaps: "
print " ".join(map(str,arr))
```

C#

```
 // Below is C# code to print largest
// permutation after atmost K swaps.
using System;

class GFG {

    // Function to calculate largest
    // permutation after atmost K
    // swaps
    static void KswapPermutation(int []arr,
                                int n, int k)
    {

        // Auxiliary dictionary of storing
        // the position of elements
        int []pos = new int[n+1];

        for (int i = 0; i < n; ++i)
            pos[arr[i]] = i;

        for (int i = 0; i < n && k > 0; ++i)
        {

            // If element is already i'th
            // largest, then no need to swap
            if (arr[i] == n-i)
                continue;
            // Find position of i'th largest
            // value, n-i
            int temp = pos[n-i];

            // Swap the elements position
            pos[arr[i]] = pos[n-i];
            pos[n-i] = i;

            // Swap the ith largest value with
            // the current value at ith place
            int tmp1 = arr[temp];
            arr[temp] = arr[i];
            arr[i] = tmp1;

            // decrement number of swaps
            --k;
        }
    }

    // Driver method
    public static void Main()
    {

        int []arr = {4, 5, 2, 1, 3};
        int n = arr.Length;
        int k = 3;

        KswapPermutation(arr, n, k);

        Console.Write("Largest permutation "
                + "after " + k + " swaps:\n");

        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed nitin mittal.
```

```
Output:
Largest permutation after 3 swaps:
5 4 3 2 1
```

Time complexity: O(n)
Auxiliary space: O(n)

Improved By : nitin mittal

Source

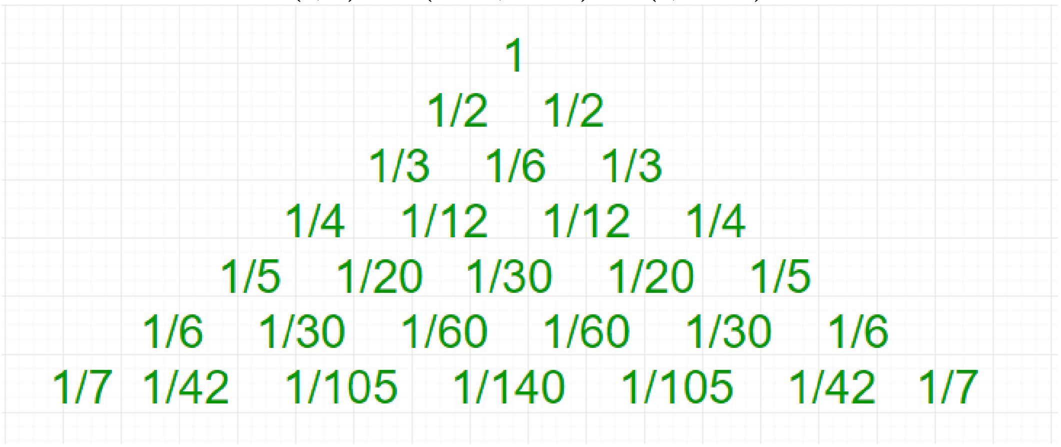
https://www.geeksforgeeks.org/largest-permutation-k-swaps/

Chapter 64

Leibniz harmonic triangle

Leibniz harmonic triangle - GeeksforGeeks

The **Leibniz harmonic triangle** is a triangular arrangement of unit fractions in which the outermost diagonals consist of the reciprocals of the row numbers and each inner cell is the cell diagonally above and to the left minus the cell to the left. To put it algebraically, $\mathbf{L(r,\ 1) = 1/r}$, where r is the number of the row, starting from 1, and c is the number, never more than r and $\mathbf{L(r,\ c) = L(r - 1,\ c - 1) - L(r,\ c - 1)}$



```
                    1
                 1/2   1/2
              1/3   1/6   1/3
          1/4   1/12   1/12   1/4
       1/5   1/20   1/30   1/20   1/5
    1/6   1/30   1/60   1/60   1/30   1/6
1/7  1/42   1/105   1/140   1/105   1/42   1/7
```

Relation with pascal's triangle
Whereas each entry in Pascal's triangle is the sum of the two entries in the above row, each entry in the Leibniz triangle is the sum of the two entries in the row below it. For example, in the 5th row, the entry (1/30) is the sum of the two (1/60)s in the 6th row.
Just as Pascal's triangle can be computed by using binomial coefficients, so can Leibniz's:

[illegible]

Properties
If one takes the denominators of the nth row and adds them, then the result will equal $\mathbf{n.2^{n\text{-}1}}$. For example, for the 3rd row, we have $3 + 6 + 3 = 12 = 3 \times 22$.

Given a positive integer **n**. The task is to print **Leibniz harmonic triangle** of height n.

Examples:

```
Input : n = 4
Output :
1
1/2 1/2
1/3 1/6 1/3
1/4 1/12 1/12 1/4

Input : n = 3
Output :
1
1/2 1/2
1/3 1/6 1/3
```

Below is the implementation of printing Leibniz harmonic triangle of height n based on above relation with Pascal triangle.

C++

```
 // CPP Program to print Leibniz Harmonic Triangle
#include <bits/stdc++.h>
using namespace std;

// Print Leibniz Harmonic Triangle
void LeibnizHarmonicTriangle(int n)
{
    int C[n + 1][n + 1];

    // Calculate value of Binomial Coefficient in
    // bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, n); j++) {

            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }
    // printing Leibniz Harmonic Triangle
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++)
            cout << "1/" << i * C[i - 1][j - 1] << " ";

        cout << endl;
    }
}

// Driven Program
int main()
{
    int n = 4;
    LeibnizHarmonicTriangle(n);
    return 0;
}
```

Java

```
 // Java Program to print
// Leibniz Harmonic Triangle
import java.io.*;
import java.math.*;

class GFG {

    // Print Leibniz Harmonic Triangle
    static void LeibnizHarmonicTriangle(int n)
    {
        int C[][] = new int[n + 1][n + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= Math.min(i, n);
                                              j++) {

                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1] +
                              C[i - 1][j];
            }
        }

        // printing Leibniz Harmonic Triangle
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
                System.out.print("1/" + i * C[i - 1][j - 1]
                                         + " ");

            System.out.println();
        }
    }

    // Driven Program
    public static void main(String args[])
    {
        int n = 4;
        LeibnizHarmonicTriangle(n);
    }
}

// This code is contributed by Nikita Tiwari
```

Python3

```
 # Python3 Program to print
# Leibniz Harmonic Triangle

# Print Leibniz Harmonic
# Triangle
def LeibnizHarmonicTriangle(n):
    C = [[0 for x in range(n + 1)]
            for y in range(n + 1)];

    # Calculate value of Binomial
    # Coefficient in bottom up manner
    for i in range(0, n + 1):
        for j in range(0, min(i, n) + 1):

            # Base Cases
            if (j == 0 or j == i):
                C[i][j] = 1;

            # Calculate value using
            # previously stored values
            else:
                C[i][j] = (C[i - 1][j - 1] +
                           C[i - 1][j]);
    # printing Leibniz
    # Harmonic Triangle
    for i in range(1, n + 1):
        for j in range(1, i + 1):
            print("1/", end = "");
            print(i * C[i - 1][j - 1],
                           end = " ");
        print();

# Driver Code
LeibnizHarmonicTriangle(4);

# This code is contributed
# by mits.
```

C#

```
 // C# Program to print Leibniz Harmonic Triangle
using System;

class GFG {

    // Print Leibniz Harmonic Triangle
    static void LeibnizHarmonicTriangle(int n)
    {
        int [,]C = new int[n + 1,n + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= Math.Min(i, n);
                                       j++) {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i,j] = C[i - 1,j - 1] +
                             C[i - 1,j];
            }
        }

        // printing Leibniz Harmonic Triangle
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
```

```
                Console.Write("1/" + i * C[i - 1,j - 1]
                                        + " ");

        Console.WriteLine();
        }
    }

    // Driven Program
    public static void Main()
    {
        int n = 4;

        LeibnizHarmonicTriangle(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to print
// Leibniz Harmonic Triangle

// Print Leibniz Harmonic Triangle
function LeibnizHarmonicTriangle($n)
{

    // Calculate value of
    // Binomial Coefficient in
    // bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $n); $j++)
        {

            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                                 $C[$i - 1][$j];
        }
    }
    // printing Leibniz
    // Harmonic Triangle
    for ($i = 1; $i <= $n; $i++)
    {
        for ($j = 1; $j <= $i; $j++)
            echo "1/", $i * $C[$i - 1][$j - 1], " ";

        echo "\n";
    }
}

    // Driver Code
    $n = 4;
    LeibnizHarmonicTriangle($n);

// This code is contributed by aj_36
?>
```

Output:

```
1/1
1/2 1/2
1/3 1/6 1/3
1/4 1/12 1/12 1/4
```

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/leibniz-harmonic-triangle/

Chapter 65

Lexicographic rank of a string

Lexicographic rank of a string - GeeksforGeeks

Given a string, find its rank among all its permutations sorted lexicographically. For example, rank of "abc" is 1, rank of "acb" is 2, and rank of "cba" is 6.

Examples:

```
Input : str[] = "acb"
Output : Rank = 2

Input : str[] = "string"
Output : Rank = 598

Input : str[] = "cba"
Output : Rank = 6
```

For simplicity, let us assume that the string does not contain any duplicated characters.

One simple solution is to initialize rank as 1, generate all permutations in lexicographic order. After generating a permutation, check if the generated permutation is same as given string, if same, then return rank, if not, then increment the rank by 1. The time complexity of this solution will be exponential in worst case. Following is an efficient solution.

Let the given string be "STRING". In the input string, 'S' is the first character. There are total 6 characters and 4 of them are smaller than 'S'. So there can be 4 * 5! smaller strings where first character is smaller than 'S', like following

R X X X X X
I X X X X X
N X X X X X
G X X X X X

Now let us Fix S' and find the smaller strings staring with 'S'.

Repeat the same process for T, rank is 4*5! + 4*4! +…

Now fix T and repeat the same process for R, rank is 4*5! + 4*4! + 3*3! +…

Now fix R and repeat the same process for I, rank is 4*5! + 4*4! + 3*3! + 1*2! +…

Now fix I and repeat the same process for N, rank is 4*5! + 4*4! + 3*3! + 1*2! + 1*1! +…

Now fix N and repeat the same process for G, rank is 4*5! + 4*4! + 3*3! + 1*2! + 1*1! + 0*0!

Rank = 4*5! + 4*4! + 3*3! + 1*2! + 1*1! + 0*0! = 597

Note that the above computations find count of smaller strings. Therefore rank of given string is count of smaller strings plus 1. The final rank = 1 + 597 = 598

C

```
 #include <stdio.h>
#include <string.h>

// A utility function to find factorial of n
int fact(int n)
{
    return (n <= 1)? 1 :n * fact(n-1);
}

// A utility function to count smaller characters on right
// of arr[low]
int findSmallerInRight(char* str, int low, int high)
{
    int countRight = 0, i;

    for (i = low+1; i <= high; ++i)
        if (str[i] < str[low])
            ++countRight;

    return countRight;
}

// A function to find rank of a string in all permutations
// of characters
int findRank (char* str)
{
    int len = strlen(str);
    int mul = fact(len);
    int rank = 1;
    int countRight;

    int i;
    for (i = 0; i < len; ++i)
    {
        mul /= len - i;

        // count number of chars smaller than str[i]
        // fron str[i+1] to str[len-1]
        countRight = findSmallerInRight(str, i, len-1);

        rank += countRight * mul ;
    }

    return rank;
}

// Driver program to test above function
int main()
{
    char str[] = "string";
    printf ("%d", findRank(str));
    return 0;
}
```

Java

```
 // Java program to find lexicographic rank
// of a string
import java.io.*;
import java.util.*;

class GFG{

    // A utility function to find factorial of n
    static int fact(int n)
    {
        return (n <= 1)? 1 :n * fact(n-1);
    }

    // A utility function to count smaller
    // characters on right of arr[low]
    static int findSmallerInRight(String str, int low,
                                        int high)
    {
        int countRight = 0, i;

        for (i = low + 1; i <= high; ++i)
            if (str.charAt(i) < str.charAt(low))
                ++countRight;
        return countRight;
    }

    // A function to find rank of a string in
    // all permutations of characters
    static int findRank (String str)
    {
        int len = str.length();
        int mul = fact(len);
        int rank = 1;
        int countRight;

        for (int i = 0; i < len; ++i)
        {
            mul /= len - i;

            // count number of chars smaller
            // than str[i] from str[i+1] to
            // str[len-1]
            countRight = findSmallerInRight(str, i, len-1);

            rank += countRight * mul ;
        }

        return rank;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        String str = "string";
        System.out.println (findRank(str));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python

```
 # Python program to find lexicographic
# rank of a string

# A utility function to find factorial
# of n
def fact(n) :
    f = 1
    while n >= 1 :
        f = f * n
        n = n - 1
    return f

# A utility function to count smaller
# characters on right of arr[low]
def findSmallerInRight(st, low, high) :

    countRight = 0
    i = low + 1
    while i <= high :
        if st[i] < st[low] :
            countRight = countRight + 1
        i = i + 1

    return countRight

# A function to find rank of a string
# in all permutations of characters
def findRank (st) :
    ln = len(st)
    mul = fact(ln)
    rank = 1
    i = 0

    while i < ln :

        mul = mul / (ln - i)

        # count number of chars smaller
        # than str[i] fron str[i+1] to
        # str[len-1]
        countRight = findSmallerInRight(st, i, ln-1)

        rank = rank + countRight * mul
        i = i + 1

    return rank


# Driver program to test above function
st = "string"
print (findRank(st))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find lexicographic rank
```

```
// of a string
using System;

class GFG{

    // A utility function to find factorial of n
    static int fact(int n)
    {
        return (n <= 1)? 1 :n * fact(n-1);
    }

    // A utility function to count smaller
    // characters on right of arr[low]
    static int findSmallerInRight(string str,
                           int low, int high)
    {
        int countRight = 0, i;

        for (i = low + 1; i <= high; ++i)
            if (str[i] < str[low])
                ++countRight;

        return countRight;
    }

    // A function to find rank of a string in
    // all permutations of characters
    static int findRank (string str)
    {
        int len = str.Length;
        int mul = fact(len);
        int rank = 1;
        int countRight;

        for (int i = 0; i < len; ++i)
        {
            mul /= len - i;

            // count number of chars smaller
            // than str[i] from str[i+1] to
            // str[len-1]
            countRight = findSmallerInRight(str,
                                  i, len-1);

            rank += countRight * mul ;
        }

        return rank;
    }

    // Driver program to test above function
    public static void Main()
    {
        string str = "string";
        Console.Write (findRank(str));
    }
}

// This code is contributed nitin mittal.
```

Output:

```
598
```

The time complexity of the above solution is O(n^2). We can reduce the time complexity to O(n) by creating an auxiliary array of size 256. See following code.

```
 // A O(n) solution for finding rank of string
#include <stdio.h>
#include <string.h>
#define MAX_CHAR 256

// A utility function to find factorial of n
int fact(int n)
{
    return (n <= 1)? 1 :n * fact(n-1);
}

// Construct a count array where value at every index
// contains count of smaller characters in whole string
void populateAndIncreaseCount (int* count, char* str)
{
    int i;

    for( i = 0; str[i]; ++i )
        ++count[ str[i] ];

    for( i = 1; i < MAX_CHAR; ++i )
        count[i] += count[i-1];
}

// Removes a character ch from count[] array
// constructed by populateAndIncreaseCount()
void updatecount (int* count, char ch)
{
    int i;
    for( i = ch; i < MAX_CHAR; ++i )
        --count[i];
}

// A function to find rank of a string in all permutations
// of characters
int findRank(char* str)
{
    int len = strlen(str);
    int mul = fact(len);
    int rank = 1, i;

    // all elements of count[] are initialized with 0
    int count[MAX_CHAR] = {0};

    // Populate the count array such that count[i]
    // contains count of characters which are present
    // in str and are smaller than i
    populateAndIncreaseCount( count, str );

    for (i = 0; i < len; ++i)
    {
        mul /= len - i;

        // count number of chars smaller than str[i]
        // fron str[i+1] to str[len-1]
        rank += count[ str[i] - 1] * mul;

        // Reduce count of characters greater than str[i]
        updatecount (count, str[i]);
    }

    return rank;
}

// Driver program to test above function
int main()
{
    char str[] = "string";
    printf ("%d", findRank(str));
    return 0;
}
```

The above programs don't work for duplicate characters. To make them work for duplicate characters, find all the characters that are smaller (include equal this time also), do the same as above but, this time divide the rank so formed by p! where p is the count of occurrences of the repeating character.

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/lexicographic-rank-of-a-string/

Chapter 66

Lexicographic rank of a string using STL

Lexicographic rank of a string using STL - GeeksforGeeks

You are given a string, find its rank among all its permutations sorted exicographically.

Examples:

```
Input : str[] = "acb"
Output : Rank = 2

Input : str[] = "string"
Output : Rank = 598

Input : str[] = "cba"
Output : Rank = 6
```

We have already discussed solutions to find Lexicographic rank of string

In this post, we use the STL function "next_permutation ()" to generate all possible permutations of the given string and, as it gives us permutations in lexicographic order, we will put an iterator to find the rank of each string. While iterating when Our permuted string becomes identical to the original input string, we break from the loop and the iterator value for the last iteration is our required result.

```
 // C++ program to print rank of
// string using next_permute()
#include <bits/stdc++.h>
using namespace std;

// Function to print rank of string
// using next_permute()
int findRank(string str)
{
    // store original string
    string orgStr = str;

    // Sort the string in lexicographically
    // ascending order
    sort(str.begin(), str.end());

    // Keep iterating until
    // we reach equality condition
    long int i = 1;
    do {
        // check for nth iteration
        if (str == orgStr)
            break;

        i++;
    } while (next_permutation(str.begin(), str.end()));

    // return iterator value
    return i;
}

// Driver code
int main()
{
    string str = "GEEKS";
    cout << findRank(str);
    return 0;
}
```

Output:

```
25
```

Source

https://www.geeksforgeeks.org/lexicographic-rank-string-using-stl/

Chapter 67

Lexicographically smallest permutation of {1, .. n} such that no. and position do not match

Lexicographically smallest permutation of {1, .. n} such that no. and position do not match - GeeksforGeeks

Given a positive integer n, find the lexicographically smallest permutation p of {1, 2, .. n} such that p_i != i. i.e., i should not be there at i-th position where i varies from 1 to n.

Examples:

```
Input : 5
Output : 2 1 4 5 3
Consider the two permutations that follow
the requirement that position and numbers
should not be same.
p = (2, 1, 4, 5, 3) and q = (2, 4, 1, 5, 3).
Since p is lexicographically smaller, our
output is p.

Input  : 6
Output : 2 1 4 3 6 5
```

Since we need lexicographically smallest (and 1 cannot come at position 1), we put 2 at first position. After 2, we put the next smallest element i.e., 1. After that the next smallest considering it does not violates our condition of pi != i.
Now, if our n is even we simply take two variables one which will contain our count of even

numbers and one which will contain our count of odd numbers and then we will keep them adding in the vector till we reach n.
But, if our n is odd, we do the same task till we reach n-1 because if we add till n then in the end we will end up having $p_i = i$. So when we reach n-1, we first add n to the position n-1 and then on position n we will put n-2.
The C++ implementation of the above program is given below.

```
 #include <bits/stdc++.h>
using namespace std;

// Function to print the permutation
void findPermutation(vector<int> a, int n)
{
    vector<int> res;

    // Initial numbers to be pushed to result
    int en = 2, on = 1;

    // If n is even
    if (n % 2 == 0) {
        for (int i = 0; i < n; i++) {
            if (i % 2 == 0) {
                res.push_back(en);
                en += 2;
            } else {
                res.push_back(on);
                on += 2;
            }
        }
    }

    // If n is odd
    else {
        for (int i = 0; i < n - 2; i++) {
            if (i % 2 == 0) {
                res.push_back(en);
                en += 2;
            } else {
                res.push_back(on);
                on += 2;
            }
        }
        res.push_back(n);
        res.push_back(n - 2);
    }

    // Print result
    for (int i = 0; i < n; i++)
```

```
        cout << res[i] << " ";
    cout << "\n";
}

// Driver Code
int main()
{
    long long int n = 9;
    findPermutation(n);
    return 0;
}
```

Output:

```
2 1 4 3 6 5 8 9 7
```

Source

https://www.geeksforgeeks.org/lexicographically-smallest-permutation-1-n-no-position-not-match/

Chapter 68

Lexicographically smallest permutation with distinct elements using minimum replacements

Lexicographically smallest permutation with distinct elements using minimum replacements - GeeksforGeeks

Given an array of n positive integers such that each element of integer is from 1 to n. Find the lexicographically permutation that can be obtained by replacing minimum number of elements in array such that every element of array occurs exactly one in the entire array. First print the minimum number of replacement required and then print the final lexicographical array.

```
Input arr[] = {2, 3, 4, 3, 2}
Output 2
          1 3 4 5 2
Explanation
Replace number '2' at position 1st with number
'1' and '3' at position 4th with number '5'.
The array that we obtain is [1, 3, 4, 5, 2]
which is lexicographically smallest among all
the possible suitable.

Input arr[] = {2, 1, 2, 1, 2}
Output 3
          2 1 3 4 5
```

Naive approach is to generate all the permutation from 1 to n and pick the smallest one which renders the minimum replacements. Time complexity of this approach is O(n!) which will definitely time out for large value of n.

Efficient approach is to pick desired elements greedily. Firstly initialize the cnt[] array which will contains the frequency of elements occuring in the array. For each element of array(a_i), occurred more than once in an array, add the numbers in ascending order because of getting lexicographically minimal permutation. For instance,
Iterate the array over all elements. Let the current number of array is a_i. If count of a_i is equals to 1 then move to next number of array. If count of a_i is greater than 1 then replace the number a_i with element **ele**(smallest number which is not occurred in array) only if **ele** $< \mathbf{a_i}$. Meanwhile decrease the count of a_i in cnt[] array.
If ele $> a_i$ then mark the number a_i so that we can replace it in next iteration. *This step this need because we need to make smallest lexicographically permutation.*

> For example, let's suppose the arr[] = {1, 5, 4, 5, 3, 7, 3}
> **In first iteration** '5' occurs two times in array(indexing 1),
> therefore we have to replace '5' at position '2' with '2'(2 < 5).
> Now the updated array = {1, 2, 4, 5, 3, 7, 3}
>
> **In next iteration**, '3' would be consider as it occurs two times
> in array. But this time the next element of replacement would
> be equals to 6 which is greater than 3. Therefore visit element
> 3 in boolean array vis[] and iterate over other elements.
> Now again '3' occurred at position 7th, this time replace it with
> number '6'.
> Final array is arr[] = {1, 2, 4, 5, 3, 7, 6}

```
 // C++ program to print lexicographically
// permutation array by replacing minimum
// element of array
#include <bits/stdc++.h>
using namespace std;

// Function to calculate lexicographically permutation
// in array
void lexicoSmallestPermuatation(int arr[], int n)
{
    // Calculate frequency of array elements
    int cnt[n + 1];
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < n; ++i)
        ++cnt[arr[i]];

    int ele = 1, replacement = 0;
    bool vis[n + 1];
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i < n; ++i) {

        // If count of element is 1, no
        // need to replace
        if (cnt[arr[i]] == 1)
            continue;

        // Find the element that has not
        // occurred in array
        while (cnt[ele])
            ++ele;

        // If replacement element is greater
        // than current arr[i] then visit
        // that element for next iteration
        if (ele > arr[i] && !vis[arr[i]])
            vis[arr[i]] = 1;

        else {

            // Decrement count and assign the element
            // to array
            --cnt[arr[i]];
            arr[i] = ele;

            // Increment the replacement count
            ++replacement;

            // Increment element after assigning
            // to the array
            ++ele;
        }
    }

    cout << replacement << "\n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 4, 3, 2 };
    int sz = sizeof(arr) / sizeof(arr[0]);
    lexicoSmallestPermuatation(arr, sz);
    return 0;
}
```

Output

```
2
1 3 4 5 2
```

Time complexity: O(n)
Auxiliary space: O()

Source

https://www.geeksforgeeks.org/lexicographically-smallest-permutation-distinct-elements-using-minimum-replaceme

Chapter 69

Lobb Number

Lobb Number - GeeksforGeeks

In combinatorial mathematics, the Lobb number $\mathbf{L_{m,\ n}}$ counts the number of ways that n + m open parentheses can be arranged to form the start of a valid sequence of balanced parentheses.
The Lobb number are parameterized by two non-negative integers m and n with n >= m >= 0. It can be obtained by:

$$L_{m,n} = \frac{2m+1}{m+n+1}\binom{2n}{m+n}$$

Lobb Number is also used to count the number of ways in which n + m copies of the value +1 and n − m copies of the value -1 may be arranged into a sequence such that all of the partial sums of the sequence are non- negative.

Examples :

```
Input : n = 3, m = 2
Output : 5

Input : n =5, m =3
Output :35
```

The idea is simple, we use a function that computes binomial coefficients for given values. Using this function and above formula, we can compute Lobb numbers.

C++

```
 // CPP Program to find Ln, m Lobb Number.
#include <bits/stdc++.h>
#define MAXN 109
using namespace std;
```

```
// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int C[n + 1][k + 1];

    // Calculate value of Binomial Coefficient in
    // bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, k); j++) {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    return C[n][k];
}

// Return the Lm, n Lobb Number.
int lobb(int n, int m)
{
    return ((2 * m + 1) * binomialCoeff(2 * n, m + n)) / (m + n + 1);
}

// Driven Program
int main()
{
    int n = 5, m = 3;
    cout << lobb(n, m) << endl;
    return 0;
}
```

Java

```
 // JAVA Code For Lobb Number
import java.util.*;

class GFG {

    // Returns value of Binomial
    // Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int C[][] = new int[n + 1][k + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= Math.min(i, k);
                                        j++) {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1] +
                              C[i - 1][j];
            }
        }

        return C[n][k];
    }

    // Return the Lm, n Lobb Number.
    static int lobb(int n, int m)
    {
        return ((2 * m + 1) * binomialCoeff(2 * n, m + n)) /
                                             (m + n + 1);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 5, m = 3;
        System.out.println(lobb(n, m));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python 3

```
 # Python 3 Program to find Ln,
# m Lobb Number.

# Returns value of Binomial
# Coefficient C(n, k)
def binomialCoeff(n, k):
    C = [[0 for j in range(k + 1)]
             for i in range(n + 1)]


    # Calculate value of Binomial
    # Coefficient in bottom up manner
    for i in range(0, n + 1):
        for j in range(0, min(i, k) + 1):
            # Base Cases
            if (j == 0 or j == i):
                C[i][j] = 1

            # Calculate value using
            # previously stored values
            else:
                C[i][j] = (C[i - 1][j - 1]
                            + C[i - 1][j])

    return C[n][k]

# Return the Lm, n Lobb Number.
def lobb(n, m):

    return (((2 * m + 1) *
        binomialCoeff(2 * n, m + n))
                      / (m + n + 1))

# Driven Program
n = 5
m = 3
print(int(lobb(n, m)))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Code For Lobb Number
using System;

class GFG {

    // Returns value of Binomial
    // Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int[, ] C = new int[n + 1, k + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= Math.Min(i, k);
                j++) {

                // Base Cases
                if (j == 0 || j == i)
                    C[i, j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i, j] = C[i - 1, j - 1]
                                + C[i - 1, j];
            }
        }

        return C[n, k];
    }

    // Return the Lm, n Lobb Number.
    static int lobb(int n, int m)
    {
        return ((2 * m + 1) * binomialCoeff(
                 2 * n, m + n)) / (m + n + 1);
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 5, m = 3;

        Console.WriteLine(lobb(n, m));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find Ln,
// m Lobb Number.

$MAXN =109;
```

```
// Returns value of Binomial
// Coefficient C(n, k)
function binomialCoeff($n, $k)
{
    $C= array(array());

    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $k); $j++)
        {
            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using p
            // reviously stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    return $C[$n][$k];
}

// Return the Lm, n Lobb Number.
function lobb($n, int $m)
{
    return ((2 * $m + 1) *
             binomialCoeff(2 * $n, $m + $n)) /
                          ($m + $n + 1);
}

// Driven Code
$n = 5;$m = 3;
echo lobb($n, $m);

// This code is contributed by anuj_67.
?>
```

Output :

```
35
```

Improved By : Smitha Dinesh Semwal, vt_m, RajatRaj

Source

https://www.geeksforgeeks.org/lobb-number/

Chapter 70

Longest common subsequence with permutations allowed

Longest common subsequence with permutations allowed - GeeksforGeeks

Given two strings in lowercase, find the longest string whose permutations are subsequences of given two strings. The output longest string must be sorted.

Examples:

```
Input  :  str1 = "pink", str2 = "kite"
Output : "ik"
The string "ik" is the longest sorted string
whose one permutation "ik" is subsequence of
"pink" and another permutation "ki" is
subsequence of "kite".

Input  : str1 = "working", str2 = "women"
Output : "now"

Input  : str1 = "geeks" , str2 = "cake"
Output : "ek"

Input  : str1 = "aaaa" , str2 = "baba"
Output : "aa"
```

The idea is to count characters in both strings.

1. calculate frequency of characters for each string and store them in their respective count arrays, say count1[] for str1 and count2[] for str2.
2. Now we have count arrays for 26 characters. So traverse count1[] and for any index 'i' append character ('a'+i) in resultant string 'result' by min(count1[i], count2[i]) times.

3. Since we traverse count array in ascending order, our final string characters will be in sorted order.

Source

https://www.geeksforgeeks.org/longest-common-subsequence-with-permutations-allowed/

Chapter 71

Make all combinations of size k

Make all combinations of size k - GeeksforGeeks

Given two numbers n and k and you have to find all possible combination of k numbers from 1…n.

Examples:

```
Input : n = 4
        k = 2
Output : 1 2
         1 3
         1 4
         2 3
         2 4
         3 4

Input : n = 5
        k = 3
Output : 1 2 3
         1 2 4
         1 2 5
         1 3 4
         1 3 5
         1 4 5
         2 3 4
         2 3 5
         2 4 5
         3 4 5
```

We have discussed one approach in below post.

Print all possible combinations of r elements in a given array of size n

In this, we use DFS based approach. We want all numbers from 1 to n. We first push all numbers from 1 to k in tmp_vector and as soon as k is equal to 0, we push all numbers from tmp_vector to ans_vector. After this we remove the last element from tmp_vector and make make all remaining combination.

```
 // C++ program to print all combinations of size
// k of elements in set 1..n
#include <bits/stdc++.h>
using namespace std;

void makeCombiUtil(vector<vector<int> >& ans,
    vector<int>& tmp, int n, int left, int k)
{
    // Pushing this vector to a vector of vector
    if (k == 0) {
        ans.push_back(tmp);
        return;
    }

    // i iterates from left to n. First time
    // left will be 1
    for (int i = left; i <= n; ++i)
    {
        tmp.push_back(i);
        makeCombiUtil(ans, tmp, n, i + 1, k - 1);

        // Popping out last inserted element
        // from the vector
        tmp.pop_back();
    }
}

// Prints all combinations of size k of numbers
// from 1 to n.
vector<vector<int> > makeCombi(int n, int k)
{
    vector<vector<int> > ans;
    vector<int> tmp;
    makeCombiUtil(ans, tmp, n, 1, k);
    return ans;
}

// Driver code
int main()
{
    // given number
    int n = 5;
    int k = 3;
    vector<vector<int> > ans = makeCombi(n, k);
    for (int i = 0; i < ans.size(); i++) {
        for (int j = 0; j < ans[i].size(); j++) {
            cout << ans.at(i).at(j) << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

Source

https://www.geeksforgeeks.org/make-combinations-size-k/

Chapter 72

Maximum binomial coefficient term value

Maximum binomial coefficient term value - GeeksforGeeks

Given a positive integer **n**. The task is to find the maximum coefficient term in all binomial coefficient.
The binomial coefficient series is
$\mathbf{{}^nC_0, {}^nC_1, {}^nC_2, ..., {}^nC_r, ..., {}^nC_{n-2}, {}^nC_{n-1}, {}^nC_n}$
the task is to find maximum value of $\mathbf{{}^nC_r}$.

Examples:

```
Input : n = 4
Output : 6
4C0 = 1
4C1 = 4
4C2 = 6
4C3 = 1
4C4 = 1
So, maximum coefficient value is 6.

Input : n = 3
Output : 3
```

Method 1: (Brute Force)
The idea is to find all the value of binomial coefficient series and find the maximum value in the series.

Below is the implementation of this approach:

C++

```
 // CPP Program to find maximum binomial coefficient
// term
#include<bits/stdc++.h>
using namespace std;

// Return maximum binomial coefficient term value.
int maxcoefficientvalue(int n)
{
    int C[n+1][n+1];

    // Calculate value of Binomial Coefficient in
    // bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, n); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }

    // finding the maximum value.
    int maxvalue = 0;
    for (int i = 0; i <= n; i++)
        maxvalue = max(maxvalue, C[n][i]);

    return maxvalue;
}

// Driven Program
int main()
{
    int n = 4;
    cout << maxcoefficientvalue(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to find
// maximum binomial
// coefficient term
```

```
import java.io.*;

class GFG
{
// Return maximum binomial
// coefficient term value.
static int maxcoefficientvalue(int n)
{
    int [][]C = new int[n + 1][n + 1];

    // Calculate value of
    // Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0;
                 j <= Math.min(i, n); j++)
        {

            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] +
                          C[i - 1][j];
        }
    }

    // finding the
    // maximum value.
    int maxvalue = 0;

    for (int i = 0; i <= n; i++)
        maxvalue = Math.max(maxvalue, C[n][i]);

    return maxvalue;
}

// Driver Code
public static void main (String[] args)
{
    int n = 4;
    System.out.println(maxcoefficientvalue(n));
}
}

// This code is contributed by ajit
```

Python3

```
 # Python3 Program to find
# maximum binomial
# coefficient term

# Return maximum binomial
# coefficient term value.
def maxcoefficientvalue(n):
    C = [[0 for x in range(n + 1)]
            for y in range(n + 1)];

    # Calculate value of
    # Binomial Coefficient in
    # bottom up manner
    for i in range(n + 1):
        for j in range(min(i, n) + 1):

            # Base Cases
            if (j == 0 or j == i):
                C[i][j] = 1;

            # Calculate value
            # using previously
            # stored values
            else:
                C[i][j] = (C[i - 1][j - 1] +
                           C[i - 1][j]);

    # finding the maximum value.
    maxvalue = 0;
    for i in range(n + 1):
        maxvalue = max(maxvalue, C[n][i]);

    return maxvalue;

# Driver Code
n = 4;
print(maxcoefficientvalue(n));

# This code is contributed by mits
```

C#

```
 // C# Program to find maximum binomial coefficient
// term
using System;

public class GFG {

    // Return maximum binomial coefficient term value.
    static int maxcoefficientvalue(int n)
    {
        int [,]C = new int[n+1,n+1];

        // Calculate value of Binomial Coefficient in
        // bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= Math.Min(i, n); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i,j] = C[i-1,j-1] + C[i-1,j];
            }
        }

        // finding the maximum value.
        int maxvalue = 0;

        for (int i = 0; i <= n; i++)
            maxvalue = Math.Max(maxvalue, C[n,i]);

        return maxvalue;
    }

    // Driven Program

    static public void Main ()
    {

        int n = 4;

        Console.WriteLine(maxcoefficientvalue(n));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

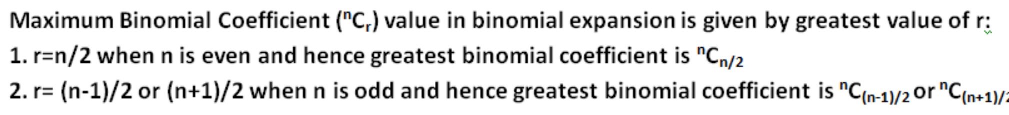
```
 <?php
// PHP Program to find
// maximum binomial
// coefficient term

// Return maximum binomial
// coefficient term value.
function maxcoefficientvalue($n)
{

    // Calculate value of
    // Binomial Coefficient in
    // bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $n); $j++)
        {

            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    // finding the maximum value.
    $maxvalue = 0;
    for ($i = 0; $i <= $n; $i++)
        $maxvalue = max($maxvalue, $C[$n][$i]);

    return $maxvalue;
}

    // Driver Code
    $n = 4;
    echo maxcoefficientvalue($n), "\n";
```

```
// This code is contributed by aj_36
?>
```

Output:

```
6
```

Method 2: (Using formula)



Maximum Binomial Coefficient (^nC_r) value in binomial expansion is given by greatest value of r:
1. r=n/2 when n is even and hence greatest binomial coefficient is $^nC_{n/2}$
2. r= (n-1)/2 or (n+1)/2 when n is odd and hence greatest binomial coefficient is $^nC_{(n-1)/2}$ or $^nC_{(n+1)/2}$

Proof,

> Expansion of $(x + y)^n$ are:
> $^nC_0\ x^n\ y^0$, $^nC_1\ x^{n-1}\ y^1$, $^nC_2\ x^{n-2}\ y^2$, …., $^nC_r\ x^{n-r}\ y^r$, …., $^nC_{n-2}\ x^2\ y^{n-2}$, $^nC_{n-1}\ x^1\ y^{n-1}$, $^nC_n\ x^0\ y^n$
>
> So, putting x = 1 and y = 1, we get binomial coefficient,
> $^nC_0$, $^nC_1$, $^nC_2$, …., ^nC_r, …., $^nC_{n-2}$, $^nC_{n-1}$, ^nC_n
>
> Let term t_{i+1} contains the greatest value in $(x + y)^n$. Therefore,
> $t_{r+1} >= t_r$
> $^nC_r\ x^{n-r}\ y^r >= {}^nC_{r-1}\ x^{n-r+1}\ y^{r-1}$
>
> Putting x = 1 and y = 1,
> $^nC_r >= {}^nC_{r-1}$
> $^nC_r/^nC_{r-1} >= 1$
> (using $^nC_r/^nC_{r-1} = (n-r+1)/r$)
> $(n-r+1)/r >= 1$
> $(n+1)/r - 1 >= 1$
> $(n+1)/r >= 2$
> $(n+1)/2 >= r$
>
> Therefore, r should be less than equal to (n+1)/2.
> And r should be integer. So, we get maximum coefficient for r equals to:
> (1) n/2, when n is even.
> (2) (n+1)/2 or (n-1)/2, when n is odd.

C++

```
 // CPP Program to find maximum binomial coefficient term
#include<bits/stdc++.h>
using namespace std;

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
```

```
{
    int C[n+1][k+1];

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, k); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }

    return C[n][k];
}

// Return maximum binomial coefficient term value.
int maxcoefficientvalue(int n)
{
    // if n is even
    if (n%2 == 0)
        return binomialCoeff(n, n/2);

    // if n is odd
    else
        return binomialCoeff(n, (n+1)/2);
}

// Driven Program
int main()
{
    int n = 4;
    cout << maxcoefficientvalue(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to find
// maximum binomial
// coefficient term
```

```
import java.io.*;

class GFG
{

    // Returns value of
    // Binomial Coefficient
    // C(n, k)
    static int binomialCoeff(int n,
                             int k)
    {
        int [][]C = new int[n + 1][k + 1];

        // Calculate value of
        // Binomial Coefficient
        // in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0;
                j <= Math.min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1] +
                              C[i - 1][j];
            }
        }
        return C[n][k];
    }

    // Return maximum
    // binomial coefficient
    // term value.
    static int maxcoefficientvalue(int n)
    {

        // if n is even
        if (n % 2 == 0)
            return binomialCoeff(n, n / 2);

        // if n is odd
        else
```

```
            return binomialCoeff(n, (n + 1) / 2);
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 4;

        System.out.println(maxcoefficientvalue(n));
    }
}

// This code is contributed
// by akt_mit
```

C#

```
 // C# Program to find maximum binomial
// coefficient term
using System;

public class GFG {

    // Returns value of Binomial Coefficient
    // C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int [,]C = new int[n+1,k+1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0;
                  j <= Math.Min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i,j] = C[i-1,j-1] +
                              C[i-1,j];
            }
        }
        return C[n,k];
    }

    // Return maximum binomial coefficient
    // term value.
    static int maxcoefficientvalue(int n)
    {

        // if n is even
        if (n % 2 == 0)
            return binomialCoeff(n, n/2);

        // if n is odd
        else
            return binomialCoeff(n, (n + 1) / 2);
    }

    // Driven Program
    static public void Main ()
    {

        int n = 4;

        Console.WriteLine(maxcoefficientvalue(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// maximum binomial
// coefficient term
// Returns value of
// Binomial Coefficient C(n, k)
function binomialCoeff($n, $k)
{
    $C[$n + 1][$k + 1] = array(0);

    // Calculate value of
    // Binomial Coefficient
    // in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0;
             $j <= min($i, $k); $j++)
        {
            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    return $C[$n][$k];
}

// Return maximum binomial
// coefficient term value.
function maxcoefficientvalue($n)
{
    // if n is even
    if ($n % 2 == 0)
        return binomialCoeff($n, $n / 2);

    // if n is odd
    else
        return binomialCoeff($n,
                            ($n + 1) / 2);
}

// Driver Code
$n = 4;
echo maxcoefficientvalue($n), "\n";

// This code is contributed by m_kit
?>
```

Output:

```
6
```

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/maximum-binomial-coefficient-term-value/

Chapter 73

Maximum points of intersection n circles

Maximum points of intersection n circles - GeeksforGeeks

Given a number n, we need to find the maximum number of times n circles intersect.

Examples:

```
Input :  n = 2
Output : 2

Input :  n = 3
Output : 6
```

As we can see in above diagram, for each pair of circles, there can be maximum two intersectuib points. Therefore if we have n circles then there can be ${}^{n}C_2$ pairs of circles in which each pair will have two intersections. So by this we can conclude that by looking at all possible pairs of circles the mathematical formula can be made for the maximum number of intersection by n circles is given by $\mathbf{2 * {}^{n}C_2}$.

$2 * {}^{n}C_2 = 2 * n * (n - 1)/2 =$ **n * (n-1)**

C++

```
 // CPP program to find maximum umber of
// intersections of n circles
#include <bits/stdc++.h>
using namespace std;

// Returns maximum number of intersections
```

```
int intersection(int n)
{
   return n * (n - 1);
}

int main()
{
    cout << intersection(3) << endl;
    return 0;
}
// This code is contributed by
// Manish Kumar Rai.
```

Java

```
 // Java program to find maximum umber of
// intersections of n circles
import java.io.*;

public class GFG {

    // for the calculation of 2*(nC2)
    static int intersection(int n)
    {
       return n * (n - 1);
    }

    public static void main(String[] args) throws IOException
    {
        System.out.println(intersection(3));
    }
}
// This code is contributed by
// Manish Kumar Rai
```

Python3

```
 # python program to find maximum umber of
# intersections of n circles
# Returns maximum number of intersections
def intersection(n):

   return n * (n - 1);

# Drive code
print(intersection(3))

# This code is contributed by Sam007
```

C#

```
 // C# program to find maximum umber of
// intersections of n circles
using System;
class GFG {

    // for the calculation of 2*(nC2)
    static int intersection(int n)
    {
        return n * (n - 1);
    }

// Driver Code
public static void Main()
{
    Console.WriteLine(intersection(3));
}

}

// This code is contributed by Sam007
```

php

```
 <?php
// php program to find maximum umber of
// intersections of n circles

// Returns maximum number of intersections
function intersection($n)
{
   return $n * ($n - 1);
}

// Drive code

echo intersection(3);


// This code is contributed by Sam007
?>
```

Output:

```
6
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/maximum-points-intersection-n-circles/

Chapter 74

Maximum sum of difference of adjacent elements

Maximum sum of difference of adjacent elements - GeeksforGeeks

Given a number n. We have to find maximum sum of all permutations of n. The maximum sum will be sum of absolute difference of adjacent elements in array.

Examples:

```
Input : 3
Output : 3
Permutations of size 3 are:
{1, 2, 3} = 1 + 1
{1, 3, 2} = 2 + 1
{2, 1, 3} = 1 + 2
{2, 3, 1} = 1 + 2
{3, 1, 2} = 2 + 1
{3, 2, 1} = 1 + 1

Input : 2
Output : 1
Permutations of 2 are:
{1, 2} = 1
{2, 1} = 1
```

Let us take example of n = 5. We can easily see we can place numbers like 1 5 2 4 3.
abs(1-5) = 4
abs(5-2) = 3
abs(2-4) = 2
abs(4-3) = 1
which sum is 4 + 3 + 2 + 1 = 10.

In general sum of this permutation is n(n-1)/2.
But the maximum sum is obtained if we move 3 at beginning of this permutation ie 3 1 5 2 4.
Sum will become 2 + 4 + 3 + 2 = 12.
We can observe that final relation will become n(n-1)/2 – 1 + n/2 for n > 1.

The permutation of n having maximum sum will be of from n/2, n-1, 2, n-2, 3, n-3. So we have to find sum of this permutation which will be n(n-1)/2 – 1 + n/2.

C++

```
 // CPP program to find maximum sum of
// adjacent elements of permutation of n
#include <iostream>
using namespace std;

// To find max sum of permutation
int maxSum(int n)
{
    // Base case
    if (n == 1)
        return 1;

    // Otherwise max sum will
    // be (n*(n-1)/2) - 1 + n/2
    else
        return (n * (n - 1) / 2) - 1 + n / 2;
}

// Driver program to test maxSum()
int main()
{
    int n = 3;
    cout << maxSum(n);
    return 0;
}
```

Java

```
 // Java program to find maximum sum of
// adjacent elements of permutaion of n
public class Main {

    // To find max sum of permutation
    static int maxSum(int n)
    {
        // Base case
        if (n == 1)
```

```
            return 1;

        // Otherwise max sum will
        // be (n*(n-1)/2) - 1 + n/2
        else
            return (n * (n - 1) / 2) - 1 + n / 2;
    }

    // Driver program to test maxSum()
    public static void main(String[] args)
    {
        int n = 3;
        System.out.println(maxSum(n));
    }
}
```

Python 3

```
 # Python program to find maximum sum of
# adjacent elements of permutation of n

# To find max sum of permutation
def maxSum(n):

    # Base case
    if (n == 1):
        return 1

    # Otherwise max sum will
    # be (n*(n-1)/2) - 1 + n / 2
    else:
        return int((n * (n - 1) / 2) - 1 + n / 2)

# Driver program to test maxSum()
n = 3
print(maxSum(n))

# This code is contributed
# by Azkia Anam.
```

C#

```
 // C# program to find maximum sum of
// adjacent elements of permutaion of n
using System;

public class main {
    // To find max sum of permutation
    static int maxSum(int n)
    {

        // Base case
        if (n == 1)
            return 1;

        // Otherwise max sum will
        // be (n*(n-1)/2) - 1 + n/2
        else
            return (n * (n - 1) / 2)
                          - 1 + n / 2;
    }

    // Driver program to test maxSum()
    public static void Main()
    {
        int n = 3;

        Console.WriteLine(maxSum(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find maximum sum of
// adjacent elements of permutation of n
// To find max sum of permutation
function maxSum( $n)
{
    // Base case
    if ($n == 1)
        return 1;

    // Otherwise max sum will
    // be (n*(n-1)/2) - 1 + n/2
    else
        return ($n * ($n - 1) / 2) -
                        1 + $n / 2;
}

    // Driver Code
    $n = 3;
    echo intval( maxSum($n));

// This code is contributed by akur
?>
```

Output:

```
3
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/maximum-sum-difference-adjacent-elements/

Chapter 75

Minimum number of operations to convert a given sequence into a Geometric Progression

Minimum number of operations to convert a given sequence into a Geometric Progression - GeeksforGeeks

Given a sequence of N elements, only three operations can be performed on any element at most one time. The operations are:

1. Add one to the element.
2. Subtract one from the element.
3. Leave the element unchanged.

Perform any one of the operations on all elements in the array. The task is to find the minimum number of operations(addition and subtraction) that can be performed on the sequence, in order to convert it into a Geometric Progression. If it is not possible to generate a GP by performing the above operations, print -1.

Examples:

> **Input**: a[] = {1, 1, 4, 7, 15, 33}
> **Output**: The minimum number of operations are 4.
> Steps:
>
> 1. Keep a_1 unchanged
> 2. Add one to a_2.
> 3. Keep a_3 unchanged
> 4. Subtract one from a_4.
> 5. Subtract one from a_5.
> 6. Add one to a_6.

The resultant sequence is {1, 2, 4, 8, 16, 32}

Input: a[] = {20, 15, 20, 15}
Output: -1

Approach The key observation to be made here is that any Geometric Progression is uniquely determined by only its first two elements (Since the ratio between each of the next pairs has to be the same as the ratio between this pair, consisting of the first two elements). Since only **3*3** permutations are possible. The possible combination of operations are (+1, +1), (+1, 0), (+1, -1), (-1, +1), (-1, 0), (-1, -1), (0, +1), (0, 0) and (0, -1). Using brute force all these **9** permutations and checking if they form a GP in linear time will give us the answer. The minimum of the operations which result in combinations which are in GP will be the answer.

Below is the implementation of the above approach:

```
 // C++ program to find minimum number
// of operations to convert a given
// sequence to an Geometric Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print the GP series
void construct(int n, pair<double, double> ans_pair)
{
    // Check for possibility
    if (ans_pair.first == -1) {
        cout << "Not possible";
        return;
    }
    double a1 = ans_pair.first;
    double a2 = ans_pair.second;
    double r = a2 / a1;

    cout << "The resultant sequence is:\n";
    for (int i = 1; i <= n; i++) {
        double ai = a1 * pow(r, i - 1);
        cout << ai << " ";
    }
}

// Function for getting the Arithmetic Progression
void findMinimumOperations(double* a, int n)
{
    int ans = INT_MAX;
    // The array c describes all the given set of
    // possible operations.
    int c[] = { -1, 0, 1 };
    // Size of c
    int possiblities = 3;

    // candidate answer
    int pos1 = -1, pos2 = -1;

    // loop through all the permutations of the first two
    // elements.
    for (int i = 0; i < possiblities; i++) {
        for (int j = 0; j < possiblities; j++) {

            // a1 and a2 are the candidate first two elements
            // of the possible GP.
            double a1 = a[1] + c[i];
            double a2 = a[2] + c[j];

            // temp stores the current answer, including the
            // modification of the first two elements.
            int temp = abs(a1 - a[1]) + abs(a2 - a[2]);

            if (a1 == 0 || a2 == 0)
                continue;

            // common ratio of the possible GP
            double r = a2 / a1;

            // To check if the chosen set is valid, and id yes
            // find the number of operations it takes.
            for (int pos = 3; pos <= n; pos++) {

                // ai is value of a[i] according to the assumed
                // first two elements a1, a2
                // ith element of an GP = a1*((a2-a1)^(i-1))
                double ai = a1 * pow(r, pos - 1);

                // Check for the "proposed" element to be only
                // differing by one
                if (a[pos] == ai) {
                    continue;
                }
                else if (a[pos] + 1 == ai || a[pos] - 1 == ai) {
                    temp++;
                }
                else {
                    temp = INT_MAX; // set the temporary ans
                    break; // to infinity and break
                }
            }
            // update answer
            if (temp < ans) {
                ans = temp;
                pos1 = a1;
                pos2 = a2;
            }
        }
    }
    if (ans == -1) {
        cout << "-1";
        return;
    }

    cout << "Minimum Number of Operations are " << ans << "\n";
    pair<double, double> ans_pair = { pos1, pos2 };

    // Calling function to print the sequence
    construct(n, ans_pair);
}

// Driver Code
int main()
{

    // array is 1-indexed, with a[0] = 0
    // for the sake of simplicity
    double a[] = { 0, 7, 20, 49, 125 };

    int n = sizeof(a) / sizeof(a[0]);

    // Function to print the minimum operations
    // and the sequence of elements
    findMinimumOperations(a, n - 1);
    return 0;
}
```

Output:

```
Minimum Number of Operations are 2
The resultant sequence is:
8 20 50 125
```

Time Complexity : O(9*N)

Source

https://www.geeksforgeeks.org/minimum-number-of-operations-to-convert-a-given-sequence-into-a-geometric-prog

Chapter 76

Missing Permutations in a list

Missing Permutations in a list - GeeksforGeeks

Given a list of permutation of any word. Find the missing permutation from the list of permutation.

Examples:

```
Input : Permutation_given[] = {"ABCD", "CABD", "ACDB",
              "DACB", "BCDA", "ACBD", "ADCB", "CDAB",
              "DABC", "BCAD", "CADB", "CDBA", "CBAD",
              "ABDC", "ADBC", "BDCA", "DCBA", "BACD",
              "BADC", "BDAC", "CBDA", "DCAB"};
Output : DBAC DBCA
```

1) We create a set of all given strings.
2) And one more set of all permutations.
3) Finally return difference between two sets.

```
 #include <bits/stdc++.h>
using namespace std;

void find_missing_strings(string Permutation_given[], size_t Size_Permutation_given)
{
    // vector "permutation" containing all
    // the permutation of input string
    vector<string> permutations;

    // Here we can take any string
    // from the given list and do
    // the necessary permutation
    string input = Permutation_given[0];
    permutations.push_back(input);

    // In the loop we will store
    // all the permutations of the string
    // in the vector "permutation".
    while (true) {

        string p = permutations.back();

        // Getting next permutation of input string
        next_permutation(p.begin(), p.end());
        if (p == permutations.front())
            break;

        permutations.push_back(p);
    }

    // vector containing all the
    // missing strings in permutation
    vector<string> missing;

    // given_permutations contains the
    // permutation of the input string
    set<string> given_permutations(Permutation_given,
         Permutation_given + Size_Permutation_given);

    // Through the set difference we will get
    // the missing words in vector missing
    set_difference(permutations.begin(), permutations.end(),
                             given_permutations.begin(),
                             given_permutations.end(),
                              back_inserter(missing));

    // printing all the missing string
    for (auto i = missing.begin(); i != missing.end(); ++i)
        cout << *i << endl;
}

// Driver code
int main()
{
    string Permutation_given[] = {
        "ABCD", "CABD", "ACDB", "DACB",
        "BCDA", "ACBD", "ADCB", "CDAB",
        "DABC", "BCAD", "CADB", "CDBA",
        "CBAD", "ABDC", "ADBC", "BDCA",
        "DCBA", "BACD", "BADC", "BDAC",
        "CBDA", "DCAB"
    };

    // size of permutation list
    size_t Size_Permutation_given =
                 sizeof(Permutation_given) /
                 sizeof(*Permutation_given);

    find_missing_strings(Permutation_given,
                    Size_Permutation_given);

    return 0;
}
```

Output:

```
DBAC
DBCA
```

Source

https://www.geeksforgeeks.org/missing-permutations-list/

Chapter 77

Narayana number

Narayana number - GeeksforGeeks

In combinatorics, the Narayana numbers **N(n, k)**, n = 1, 2, 3 …, 1 <= k <= n, form a triangular array of natural numbers, called Narayana triangle. It is given by :

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}.$$

Narayana numbers N(n, k) can be used to find the number of expressions containing *n-pairs* of parentheses, which are correctly matched and which contain k distinct nesting.

For instance, N(4, 2) = 6 as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()' :

```
()((()))  (())(())  (()(()))  ((()()))  ((())())  ((()))()
```

Examples:

```
Input : n = 6, k = 2
Output : 15

Input : n = 8, k = 5
Output : 490
```

Below is the implementation of finding N(n, k) :

C++

```
 // CPP program to find Narayana number N(n, k)
#include<bits/stdc++.h>
using namespace std;

// Return product of coefficient terms in formula
int productofCoefficiet(int n, int k)
{
    int C[n + 1][k + 1];

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, k); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    return C[n][k] * C[n][k - 1];
}

// Returns Narayana number N(n, k)
int findNN(int n, int k)
{
    return (productofCoefficiet(n, k)) / n;
}

// Driven Program
int main()
{
    int n = 8, k = 5;
    cout << findNN(n, k) << endl;
    return 0;
}
```

Java

```
 // Java program to find
// Narayana number N(n, k)
class GFG
```

```
{

    // Return product of coefficient
    // terms in formula
    static int productofCoefficiet(int n,
                                   int k)
    {
        int C[][] = new int[n + 1][k + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0;
                     j <= Math.min(i, k); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i][j] = C[i - 1][j - 1]
                                + C[i - 1][j];
            }
        }

        return C[n][k] * C[n][k - 1];
    }

    // Returns Narayana number N(n, k)
    static int findNN(int n, int k)
    {
        return (productofCoefficiet(n, k)) / n;
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 8, k = 5;
        System.out.println(findNN(n, k));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find Narayana number N(n, k)

# Return product of coefficient terms in formula
def productofCoefficiet(n, k):
    C = [[0 for x in range(k+1)] for y in range(n+1)]

    # Calculate value of Binomial Coefficient
    # in bottom up manner
    for i in range(0, n+1):
        for j in range(0, min(i+1,k+1)):

            # Base Cases
            if (j == 0 or j == i):
                C[i][j] = 1

            # Calculate value using previously
            # stored values
            else :
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

    return C[n][k] * C[n][k - 1]

# Returns Narayana number N(n, k)
def findNN(n, k):
    return (productofCoefficiet(n, k)) / n

# Driven Program
n = 8
k = 5
print(int(findNN(n, k)))

# This code is contributed by Prasad Kshirsagar
```

C#

```
 // C# program to find
// Narayana number N(n, k)
using System;

class GFG {

    // Return product of coefficient
    // terms in formula
    static int productofCoefficiet(int n,
                                   int k)
    {
        int[, ] C = new int[n + 1, k + 1];
        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0;
                j <= Math.Min(i, k); j++) {

                // Base Cases
                if (j == 0 || j == i)
                    C[i, j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i, j] = C[i - 1, j - 1]
                            + C[i - 1, j];
            }
        }

        return C[n, k] * C[n, k - 1];
    }

    // Returns Narayana number N(n, k)
    static int findNN(int n, int k)
    {
        return (productofCoefficiet(n, k)) / n;
    }

    // Driver code
    public static void Main()
    {
        int n = 8, k = 5;
        Console.WriteLine(findNN(n, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find
// Narayana number N(n, k)

// Return product of
// coefficient terms
// in formula
function productofCoefficiet($n, $k)
{
    $C = array(array());

    // Calculate value of
    // Binomial Coefficient
    // in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $k); $j++)
        {

            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    return $C[$n][$k] * $C[$n][$k - 1];
}

// Returns Narayana
// number N(n, k)
function findNN( $n, $k)
{
    return (productofCoefficiet($n, $k)) /$n;
}

    // Driver Program
    $n = 8;
    $k = 5;
    echo findNN($n, $k) ;

// This code is contributed by anuj_67.
?>
```

Output:

```
490
```

Improved By : vt_m, Prasad_Kshirsagar

Source

https://www.geeksforgeeks.org/narayana-number/

Chapter 78

Next greater number on the basis of precedence of digits

Next greater number on the basis of precedence of digits - GeeksforGeeks

Given a number **num** containing **n** digits. The problem is to find the next greater number using the same set of digits in **num** on the basis of the given precedence of digits. For example the precedence of digits is given as **1, 6, 4, 5, 2, 9, 8, 0, 7, 3** which simply means $\mathbf{1 < 6 < 4 < 5 < 2 < 9 < 8 < 0 < 7 < 3}$. If next greater number cannot be formed then print the original number.

Examples:

```
Input : num = "231447"
        pre[] = {1, 6, 7, 5, 2, 9, 8, 0, 4, 3}
Output : 237144
According to the precedence of digits 1 is
being considered as the smallest digit and 3
is being considered as the largest digit.

Input : num = "471"
        pre[] = {1, 6, 7, 5, 2, 9, 8, 0, 4, 3}
Output : 471
```

Approach: Following are the steps:

1. Create a **priority**[] array of size '10'. With the help of precedence array **pre**[] assign a priority number to each digit in **priority**[] where '1' is being considered as smallest priority and '10' as the highest priority.
2. Using the STL C++ next_permutation with a manually defined compare function find the next greater permutation.

```
 // C++ implementation to find the next greater number
// on the basis of precedence of digits
#include <bits/stdc++.h>

using namespace std;

#define DIGITS 10

// priority[] to store the priority of digits
// on the basis of pre[] array. Here '1' is being
// considered as the smallest priority as '10' as
// the highest priority
int priority[DIGITS];

// comparator function used for finding the
// the next greater permutation
struct compare {
  bool operator()(char x, char y) {
    return priority[x - '0'] < priority[y - '0'];
  }
};

// function to find the next greater number
// on the basis of precedence of digits
void nextGreater(char num[], int n, int pre[]) {
  memset(priority, 0, sizeof(priority));

  // variable to assgin priorities to digits
  int assign = 1;

  // assigning priorities to digits on
  // the basis of pre[]
  for (int i = 0; i < DIGITS; i++) {
    priority[pre[i]] = assign;
    assign++;
  }

  // find the next greater permutation of 'num'
  // using the compare() function
  bool a = next_permutation(num, num + n, compare());

  // if the next greater permutation does not exists
  // then store the original number back to 'num'
  // using 'pre_permutation'.
  if (a == false)
    prev_permutation(num, num + n, compare());
}
```

```
// Driver program to test above
int main() {
  char num[] = "231447";
  int n = strlen(num);
  int pre[] = {1, 6, 7, 5, 2, 9, 8, 0, 4, 3};
  nextGreater(num, n, pre);
  cout << "Next Greater: " << num;
  return 0;
}
```

Output:

```
Next Greater: 237144
```

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

Source

https://www.geeksforgeeks.org/next-greater-number-basis-precedence-digits/

Chapter 79

Number of Binary Trees for given Preorder Sequence length

Number of Binary Trees for given Preorder Sequence length - GeeksforGeeks

Count the number of Binary Tree possible for a given Preorder Sequence length n.

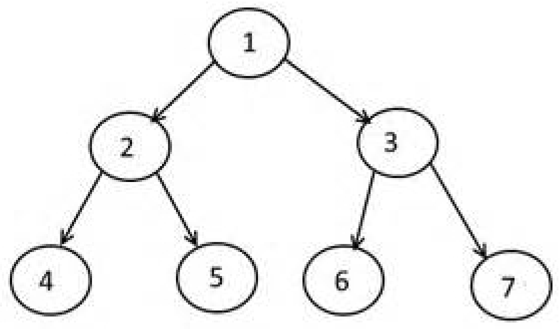
Examples:

```
Input : n = 1
Output : 1

Input : n = 2
Output : 2

Input : n = 3
Output : 5
```

Background :

In Preorder traversal, we process the root node first, then traverse the left child node and then right child node.

For example preorder traversal of below tree is 1 2 4 5 3 6 7

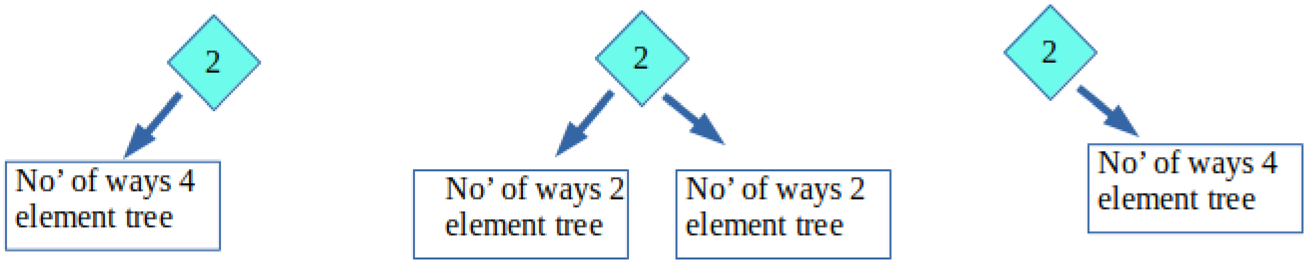


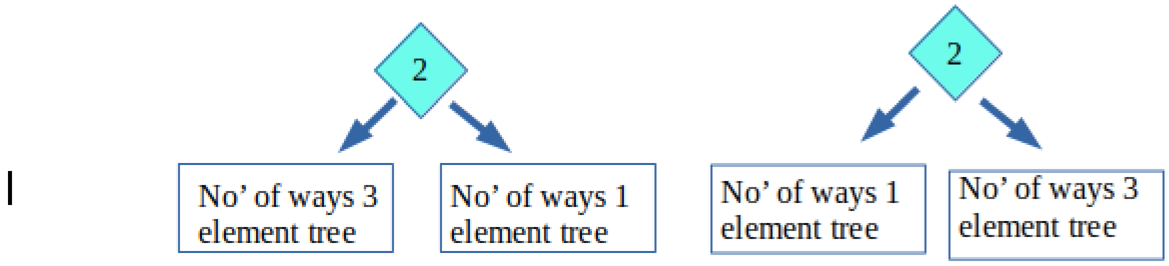
Finding number of trees with given Preorder:

Number of Binary Tree possible if such a traversal length (let's say n) is given.

Let's take an Example : Given Preorder Sequence –> 2 4 6 8 10 (length 5).

- Assume there is only 1 node (that is 2 in this case), So only 1 Binary tree is Possible
- Now, assume there are 2 nodes (namely 2 and 4), So only 2 Binary Tree are Possible:
- Now, when there are 3 nodes (namely 2, 4 and 6), So Possible Binary tree are 5
- Consider 4 nodes (that are 2, 4, 6 and 8), So Possible Binary Tree are 14.
  Let's say BT(1) denotes number of Binary tree for 1 node. (We assume BT(0)=1)
  BT(4) = BT(0) * BT(3) + BT(1) * BT(2) + BT(2) * BT(1) + BT(3) * BT(0)
  BT(4) = 1 * 5 + 1 * 2 + 2 * 1 + 5 * 1 = 14
- Similarly, considering all the 5 nodes (2, 4, 6, 8 and 10). Possible number of Binary Tree are:
  BT(5) = BT(0) * BT(4) + BT(1) * BT(3) + BT(2) * BT(2) + BT(3) * BT(1) + BT(4) * BT(0)
  BT(5) = 1 * 14 + 1 * 5 + 2 * 2 + 5 * 1 + 14 * 1 = 42


2
No' of ways 4 element tree
2
No' of ways 2 element tree
No' of ways 2 element tree
2
No' of ways 4 element tree


2
No' of ways 3 element tree
No' of ways 1 element tree
2
No' of ways 1 element tree
No' of ways 3 element tree

Hence, Total binary Tree for Pre-order sequence of length 5 is 42.

We use Dynamic programming to calculate the possible number of Binary Tree. We take one node at a time and calculate the possible Trees using previously calculated Trees.

C++

```
 // C++ Program to count possible binary trees
// using dynamic programming
#include <bits/stdc++.h>
using namespace std;

int countTrees(int n)
{
    // Array to store number of Binary tree
    // for every count of nodes
    int BT[n + 1];
    memset(BT, 0, sizeof(BT));

    BT[0] = BT[1] = 1;

    // Start finding from 2 nodes, since
    // already know for 1 node.
    for (int i = 2; i <= n; ++i)
        for (int j = 0; j < i; j++)
            BT[i] += BT[j] * BT[i - j - 1];

    return BT[n];
}

// Driver code
int main()
{
    int n = 5;
    cout << "Total Possible Binary Tree are : "
        << countTrees(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to count
// possible binary trees
// using dynamic programming
import java.io.*;

class GFG
{
static int countTrees(int n)
{
    // Array to store number
    // of Binary tree for
    // every count of nodes
    int BT[] = new int[n + 1];
    for(int i = 0; i <= n; i++)
    BT[i] = 0;
    BT[0] = BT[1] = 1;

    // Start finding from 2
    // nodes, since already
    // know for 1 node.
    for (int i = 2; i <= n; ++i)
        for (int j = 0; j < i; j++)
            BT[i] += BT[j] *
                     BT[i - j - 1];

    return BT[n];
}

// Driver code
public static void main (String[] args)
{
int n = 5;
System.out.println("Total Possible " +
                "Binary Tree are : " +
                       countTrees(n));
}
}

// This code is contributed by anuj_67.
```

Output:

```
Total Possible Binary Tree are : 42
```

Alternative :
This can also be done using Catalan number Cn = (2n)!/(n+1)!*n!

For n = 0, 1, 2, 3, … values of Catalan numbers are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, …. So are numbers of Binary Search Trees.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-of-binary-trees-for-given-preorder-sequence-length/

Chapter 80

Number of Reflexive Relations on a Set

Number of Reflexive Relations on a Set - GeeksforGeeks

Given a number n, find out number of Reflexive Relation on a set of first n natural numbers {1, 2, ..n}.

Examples :

```
Input  : n = 2
Output : 4
The given set A = {1, 2}. The following
are reflexive relations on A * A :
{{1, 1), (2, 2)}
{(1, 1), (2, 2), (1, 2)}
{(1, 1), (2, 2), (1, 2), (2, 1)}
{(1, 1), (2, 2), (2, 1)}

Input  : n = 3
Output : 64
The given set is {1, 2, 3}. There are
64 reflexive relations on A * A :
```

Explanation :

Reflexive Relation : A Relation R on A a set A is said to be Reflexive if xRx for every element of x ? A.

> The number of reflexive relations on an n-element set is $\mathbf{2^{n^2-n}}$
>
> **How does this formula work?**
> A relation R is reflexive if the matrix diagonal elements are 1.

$$\left\{\begin{matrix}1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1\end{matrix}\right\}$$

If we take a closer look the matrix, we can notice that the size of matrix is n^2. The n diagonal entries are fixed. For remaining $n^2 - n$ entries, we have choice to either fill 0 or 1. So there are total $\mathbf{2^{n^2 - n}}$ ways of filling the matrix.

CPP

```
 // C++ Program to count reflexive relations
// on a set of first n natural numbers.
#include <iostream>
using namespace std;

int countReflexive(int n)
{
   // Return 2^(n*n - n)
   return (1 << (n*n - n));
}

int main()
{
    int n = 3;
    cout << countReflexive(n);
    return 0;
}
```

Java

```
 // Java Program to count reflexive
// relations on a set of first n
// natural numbers.

import java.io.*;
import java.util.*;

class GFG {
```

```
static int countReflexive(int n)
{

// Return 2^(n*n - n)
return (1 << (n*n - n));

}

// Driver function
    public static void main (String[] args) {
    int n = 3;
    System.out.println(countReflexive(n));

    }
}

// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 Program to count
# reflexive relations
# on a set of first n
# natural numbers.


def countReflexive(n):

    # Return 2^(n*n - n)
    return (1 << (n*n - n));

# driver function
n = 3
ans = countReflexive(n);
print (ans)

# This code is contributed by saloni1297
```

C#

```
 // C# Program to count reflexive
// relations on a set of first n
// natural numbers.
using System;

class GFG {
    static int countReflexive(int n)
    {

        // Return 2^(n*n - n)
        return (1 << (n*n - n));

    }

    // Driver function
    public static void Main () {

    int n = 3;
    Console.WriteLine(countReflexive(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to count
// reflexive relations on a
// set of first n natural numbers.

function countReflexive($n)
{
// Return 2^(n * n - n)
return (1 << ($n * $n - $n));
}

//Driver code
$n = 3;
echo countReflexive($n);

// This code is contributed by mits
?>
```

Output :

```
64
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/number-reflexive-relations-set/

Chapter 81

Number of Symmetric Relations on a Set

Number of Symmetric Relations on a Set - GeeksforGeeks

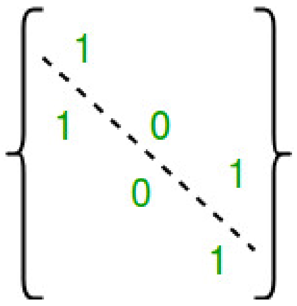
Given a number n, find out number of Symmetric Relations on a set of first n natural numbers {1, 2, ..n}.

Examples:

```
Input  : n = 2
Output : 8
Given set is {1, 2}. Below are all symmetric relation.
{}
{(1, 1)},
{(2, 2)},
{(1, 1), (2, 2)},
{(1, 2), (2, 1)}
{(1, 1), (1, 2), (2, 1)},
{(2, 2), (1, 2), (2, 1)},
{(1, 1), (1, 2), (2, 1), (1, 2)}

Input  : n = 3
Output : 64
```

A Relation 'R' on Set A is said be Symmetric if xRy then yRx for every x, y   A
or if (x, y)   R, then (y, x)   R for every x, y?A

> Total number of symmetric relations is $\mathbf{2^{n(n+1)/2}}$.
>
> **How does this formula work?**
>
> A relation R is symmetric if the value of every cell (i, j) is same as that cell (j, i). The diagonals can have any value.



There are n diagonal values, total possible combination of diagonal values = 2^n
There are $n^2 - n$ non-diagonal values. We can only choose different value for half of them, because when we choose a value for cell (i, j), cell (j, i) gets same value.
So combination of non-diagonal values = $2^{(n^2 - n)/2}$

Overall combination = $2^n * 2^{(n^2 - n)/2} = \mathbf{2^{n(n+1)/2}}$

C++

```
 // C++ program to count total symmetric relations
// on a set of natural numbers.
#include <bits/stdc++.h>

// function find the square of n
unsigned int countSymmetric(unsigned int n)
{
    // Base case
    if (n == 0)
        return 1;

   // Return 2^(n(n + 1)/2)
   return 1 << ((n * (n + 1))/2);
}

// Driver code
int main()
{
    unsigned int n = 3;

    printf("%u", countSymmetric(n));
    return 0;
}
```

Java

```
 // Java program to count total symmetric
```

```
// relations on a set of natural numbers.
import java.io.*;
import java.util.*;

class GFG {

    // function find the square of n
    static int countSymmetric(int n)
    {
        // Base case
        if (n == 0)
            return 1;

    // Return 2^(n(n + 1)/2)
    return 1 << ((n * (n + 1)) / 2);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 3;
        System.out.println(countSymmetric(n));
    }
}


// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python 3 program to count
# total symmetric relations
# on a set of natural numbers.

# function find the square of n
def countSymmetric(n) :
    # Base case
    if (n == 0) :
        return 1

    # Return 2^(n(n + 1)/2)
    return (1 << ((n * (n + 1))//2))


# Driver code

n = 3
print(countSymmetric(n))
```

```
# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program to count total symmetric
// relations on a set of natural numbers.
using System;

class GFG {

    // function find the square of n
    static int countSymmetric(int n)
    {
        // Base case
        if (n == 0)
            return 1;

    // Return 2^(n(n + 1)/2)
    return 1 << ((n * (n + 1)) / 2);
    }

    // Driver code
    public static void Main ()
    {
        int n = 3;
        Console.WriteLine(countSymmetric(n));
    }
}


// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to count total symmetric
// relations on a set of natural numbers.

// function find the square of n
function countSymmetric($n)
{
    // Base case
    if ($n == 0)
        return 1;
    // Return 2^(n(n + 1)/2)
    return 1 << (($n * ($n + 1))/2);
}

    // Driver code
    $n = 3;
    echo(countSymmetric($n));

// This code is contributed by vt_m.
?>
```

Output:

```
64
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-symmetric-relations-set/

Chapter 82

Number of arrays of size N whose elements are positive integers and sum is K

Number of arrays of size N whose elements are positive integers and sum is K - GeeksforGeeks

Given two positive integers **N** and **K**. The task is to find the number of arrays of size N that can be formed such that elements of the array should be positive integers and the sum of elements is equal to K.

Examples:

```
Input : N = 2, K = 3
Output : 2
Explanation: [1, 2] and [2, 1] are the only arrays of size 2 whose sum is 3.

Input : n = 3, k = 7
Output : 15
```

Prerequisite: Stars and Bars

Suppose there are K identical objects which needs to be placed in N bins (N indices of the array) such that each bin have at least one object. Instead of starting to place objects into bins, we start placing the objects on a line, where the object for the first bin will be taken from the left, followed by the objects for the second bin, and so forth. Thus, the configuration will be determined once one knows what is the first object going to the second bin, and the first object going to the third bin, and so on. We can indicate this by placing N X 1 separating bars at some places between two objects; since no bin is allowed to be empty, there can be at most one bar between a given pair of objects. So, we have K objects in a line with K – 1 gaps. Now we have to choose N – 1 gaps to place bars from K – 1 gaps. This can be chosen by $^{K\,-\,1}C_{N\,-\,1}$.

Below is implementation of this approach:

C++

```
 // CPP Program to find the number of arrays of
// size N whose elements are positive integers
// and sum is K
#include <bits/stdc++.h>
using namespace std;

// Return nCr
int binomialCoeff(int n, int k)
{
    int C[k + 1];
    memset(C, 0, sizeof(C));

    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++) {
        // Compute next row of pascal triangle using
        // the previous row
        for (int j = min(i, k); j > 0; j--)
            C[j] = C[j] + C[j - 1];
    }
    return C[k];
}

// Return the number of array that can be
// formed of size n and sum equals to k.
int countArray(int N, int K)
{
    return binomialCoeff(K - 1, N - 1);
}

// Driver Code
int main()
{
    int N = 2, K = 3;

    cout << countArray(N, K) << endl;

    return 0;
}
```

Java

```
 // Java Program to find the
```

```
// number of arrays of size
// N whose elements are positive
// integers and sum is K
import java.io.*;

class GFG
{

// Return nCr
static int binomialCoeff(int n,
                         int k)
{
    int []C = new int[k + 1];


    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal
        // triangle using the previous row
        for (int j = Math.min(i, k); j > 0; j--)
            C[j] = C[j] + C[j - 1];
    }
    return C[k];
}

// Return the number of
// array that can be
// formed of size n and
// sum equals to k.
static int countArray(int N, int K)
{
    return binomialCoeff(K - 1, N - 1);
}

// Driver Code
public static void main (String[] args)
{
        int N = 2, K = 3;

System.out.println( countArray(N, K));
}
}

// This code is contributed by anuj_67.
```

C#

```
 // C# Program to find the
// number of arrays of size
// N whose elements are positive
// integers and sum is K
using System;

class GFG
{
// Return nCr
static int binomialCoeff(int n,
                         int k)
{
    int []C = new int[k + 1];


    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of
        // pascal triangle using
        // the previous row
        for (int j = Math.Min(i, k);
                         j > 0; j--)
            C[j] = C[j] + C[j - 1];
    }
    return C[k];
}

// Return the number of
// array that can be
// formed of size n and
// sum equals to k.
static int countArray(int N,
                      int K)
{
    return binomialCoeff(K - 1,
                         N - 1);
}

// Driver Code
static public void Main ()
{
    int N = 2, K = 3;

    Console.WriteLine(
            countArray(N, K));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP Program to find the
// number of arrays of size
// N whose elements are
// positive integers and
// sum is K

// Return nCr
function binomialCoeff($n, $k)
{
    $C = array_fill(0, ($k + 1), 0);

    $C[0] = 1; // nC0 is 1

    for ($i = 1; $i <= $n; $i++)
    {
        // Compute next row
        // of pascal triangle
        // using the previous row
        for ($j = min($i, $k);
             $j > 0; $j--)
            $C[$j] = $C[$j] +
                     $C[$j - 1];
    }
    return $C[$k];
}

// Return the number of
// array that can be
// formed of size n and
// sum equals to k.
function countArray($N, $K)
{
    return binomialCoeff($K - 1,
                         $N - 1);
}

// Driver Code
$N = 2;
$K = 3;

echo countArray($N, $K);
```

```
// This code is contributed by mits
?>
```

Output:

```
2
```

Improved By : vt_m, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/number-arrays-size-n-whose-elements-positive-integers-sum-k/

Chapter 83

Number of bitonic arrays of length n and consisting of elements from 1 to n

Number of bitonic arrays of length n and consisting of elements from 1 to n - GeeksforGeeks

For a given number n (n > 1), we need to find the number of ways you can make bitonic array of length n, consisting of all elements from 1 to n.
Note: [1, 2,…n] and [n, n – 1…2, 1] are not considered as bitonic array.

Examples :

```
Input : n = 3
Output : 2

Explanation : [1, 3, 2] & [2, 3, 1]
are only two ways of bitonic array
formation for n = 3.

Input : n = 4
Output : 6
```

For creation of a bitonic array, let's say that we have an empty array of length n and we want to put the numbers from 1 to n in this array in bitonic form, now let's say we want to add the number 1, we have only 2 possible ways to put the number 1, both are the end positions because if we should put 1 at any place other than end points then number on both side of 1 are greater than 1. After that we can imagine that we have an array of length n-1 and now we want to put the number 2, again for the same reasons we have two ways and so on, until we want to put the number n, we will only have 1 way instead of 2, so we have n-1 numbers that have 2 ways to put, so by multiplication rule of combinatorics the answer is 2^n-1, finally we should subtract 2 from the answer because permutations 1 2 3 4 …. n and n n-1 … 3 2 1 should not be counted.

C++

```
 // C++ program for finding
// total bitonic array
#include<bits/stdc++.h>
using namespace std;

// Function to calculate no. of ways
long long int maxWays( int n)
{
    // return (2^(n - 1)) -2
    return (pow(2, n - 1) - 2);
}

// Driver Code
int main()
{
    int n = 6;
    cout << maxWays(n);
    return 0;
}
```

Java

```
 // Java program for finding
// total bitonic array
class GFG
{

    // Function to calculate no. of ways
    static int maxWays( int n)
    {

        // return (2 ^ (n - 1)) -2
        return (int)(Math.pow(2, n - 1) - 2);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 6;

        System.out.print(maxWays(n));
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
 # python program for finding
# total bitonic array

# Function to calculate no. of ways
def maxWays(n):

    # return (2^(n - 1)) -2
    return (pow(2, n - 1) - 2);

# Driver Code
n = 6;
print(maxWays(n))

# This code is contributed by Sam007
```

C#

```
 // C# program for finding
// total bitonic array
using System;

class GFG
{

    // Function to calculate no. of ways
    static int maxWays( int n)
    {

        // return (2 ^ (n - 1)) -2
        return (int)(Math.Pow(2, n - 1) - 2);
    }

    // Driver Code
    public static void Main ()
    {
        int n = 6;

        Console.Write(maxWays(n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program for finding
// total bitonic array

// Function to calculate
// no. of ways
function maxWays( $n)
{

    // return (2^(n - 1)) -2
    return (pow(2, $n - 1) - 2);
}

// Driver Code
$n = 6;
echo maxWays($n);

// This code is contributed by vt_m.
?>
```

Output:

```
30
```

Improved By : nitin mittal, vt_m, Sam007

Source

https://www.geeksforgeeks.org/number-bitonic-arrays-length-n-consisting-elements-1-n/

Chapter 84

Number of compositions of a natural number

Number of compositions of a natural number - GeeksforGeeks

Given a natural number n, find the number of ways in which n can be expressed as a sum of natural numbers when order is taken into consideration. Two sequences that differ in the order of their terms define different compositions of their sum.

Examples:

```
Input :  4
Output : 8
Explanation
All 8 position composition are:
4, 1+3, 3+1, 2+2, 1+1+2, 1+2+1, 2+1+1
and 1+1+1+1

Input :  8
Output : 128
```

A Simple Solution is to generate all compositions and count them.

Using the concept of combinatorics, it can be proved that any natural number n will have **2^(n-1) distinct compositions** when order is taken into consideration.

> One way to see why the answer is 2^(n-1) directly is to write n as a sum of 1s: n = 1 + 1 + 1 +…+ 1 (n times).
>
> There are (n-1) plus signs between all 1s. For every plus sign we can choose to split (by putting a bracket) at the point or not split. Therefore answer is 2^(n-1).

For example, n = 4
No Split
4 = 1 + 1 + 1 + 1 [We write as single 4]

Different ways to split once
4 = (1) + (1 + 1 + 1) [We write as 1 + 3]
4 = (1 + 1) + (1 + 1) [We write as 2 + 2]
4 = (1 + 1 + 1) + (1) [We write as 3 + 1]

Different ways to split twice
4 = (1) + (1 + 1) + (1) [We write as 1 + 2 + 1]
4 = (1 + 1) + (1) + (1) [We write as 2 + 1 + 1]
4 = (1) + (1) + (1 + 1) [We write as 1 + 1 + 2]

Different ways to split three times
4 = (1) + (1) + (1) + (1) [We write as 1 + 1 + 1 + 1]

Since there are **(n-1)** plus signs between the n 1s, there are 2^(n-1) ways of choosing where to split the sum, and hence 2^(n-1) possible sums .

C++

```
 // C++ program to find the total number of
// compositions of a natural number
#include<iostream>
using namespace std;

#define ull unsigned long long

ull countCompositions(ull n)
{
    // Return 2 raised to power (n-1)
    return (1L) << (n-1);
}

// Driver Code
int main()
{
    ull n = 4;
    cout << countCompositions(n) << "\n";
    return 0;
}
```

Java

```
 // Java program to find
// the total number of
// compositions of a
// natural number
import java.io.*;
import java.util.*;

class GFG
{
public static int countCompositions(int n)
{
    // Return 2 raised
    // to power (n-1)
    return 1 << (n - 1);
}

// Driver Code
public static void main(String args[])
{
    int n = 4;
    System.out.print(countCompositions(n));
}
}

// This code is contributed by
// Akanksha Rai(Abby_akku)
```

Python

```
 # Python code to find the total number of
# compositions of a natural number
def countCompositions(n):

    # function to return the total number
    # of composition of n
    return (2**(n-1))

# Driver Code
print(countCompositions(4))
```

C#

```
 // C# program to find the
// total number of compositions
// of a natural number
using System;

class GFG
{
public static int countCompositions(int n)
{
    // Return 2 raised
    // to power (n-1)
    return 1 << (n - 1);
}

// Driver Code
public static void Main()
{
    int n = 4;
    Console.Write(countCompositions(n));
}
}

// This code is contributed by mits
```

PHP

```
 <?php
// PHP program to find the
// total number of compositions
// of a natural number

function countCompositions($n)
{
    // Return 2 raised
    // to power (n-1)
    return ((1) << ($n - 1));
}

// Driver Code
$n = 4;
echo countCompositions($n), "\n";

// This code is contributed
// by ajit
?>
```

Output:

```
8
```

Improved By : jit_t, Mithun Kumar, Abby_akku

Source

https://www.geeksforgeeks.org/number-compositions-natural-number/

Chapter 85

Number of distinct permutation a String can have

Number of distinct permutation a String can have - GeeksforGeeks

We are given a string having only lowercase alphabets. The task is to find out total number of distinct permutation can be generated by that string.

Examples:

```
Input : aab
Output : 3
Different permutations are "aab",
"aba" and "baa".

Input : ybghjhbuytb
Output : 1663200
```

A **simple solution** is to find all the distinct permutation and count them.

We can find the count **without finding all permutation**. Idea is to find all the characters that is getting repeated, i.e., frequency of all the character. Then, we divide the factorial of the length of string by multiplication of factorial of frequency of characters.

In second example, number of character is 11 and here **h** and **y** are repeated **2** times whereas **g** is repeated **3** times.
So, number of permutation is **11! / (2!2!3!) = 1663200**

Below is the implementation of above idea.

C++

```
 // C++ program to find number of distinct
// permutations of a string.
#include<bits/stdc++.h>
using namespace std;
const int MAX_CHAR = 26;

// Utility function to find factorial of n.
int factorial(int n)
{
    int fact = 1;
    for (int i = 2; i <= n; i++)
        fact = fact * i;
    return fact;
}

// Returns count of distinct permutations
// of str.
int countDistinctPermutations(string str)
{
    int length = str.length();

    int freq[MAX_CHAR];
    memset(freq, 0, sizeof(freq));

    // finding frequency of all the lower case
    // alphabet and storing them in array of
    // integer
    for (int i = 0; i < length; i++)
        if (str[i] >= 'a')
            freq[str[i] - 'a']++;

    // finding factorial of number of appearances
    // and multiplying them since they are
    // repeating alphabets
    int fact = 1;
    for (int i = 0; i < MAX_CHAR; i++)
        fact = fact * factorial(freq[i]);

    // finding factorial of size of string and
    // dividing it by factorial found after
    // multiplying
    return factorial(length) / fact;
}

// Driver code
int main()
{
    string str = "fvvfhvgv";
    printf("%d", countDistinctPermutations(str));
    return 0;
}
```

Java

```
 // Java program to find number of distinct
// permutations of a string.
public class GFG {

    static final int MAX_CHAR = 26;

    // Utility function to find factorial of n.
    static int factorial(int n)
    {
        int fact = 1;
        for (int i = 2; i <= n; i++)
            fact = fact * i;
        return fact;
    }

    // Returns count of distinct permutations
    // of str.
    static int countDistinctPermutations(String str)
    {
        int length = str.length();

        int[] freq = new int[MAX_CHAR];

        // finding frequency of all the lower case
        // alphabet and storing them in array of
        // integer
        for (int i = 0; i < length; i++)
            if (str.charAt(i) >= 'a')
                freq[str.charAt(i) - 'a']++;

        // finding factorial of number of appearances
        // and multiplying them since they are
        // repeating alphabets
        int fact = 1;
        for (int i = 0; i < MAX_CHAR; i++)
            fact = fact * factorial(freq[i]);

        // finding factorial of size of string and
        // dividing it by factorial found after
        // multiplying
        return factorial(length) / fact;
    }
    // Driver code
    public static void main(String args[])
    {
        String str = "fvvfhvgv";
        System.out.println(countDistinctPermutations(str));
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Python program to find number of distinct
# permutations of a string.

MAX_CHAR = 26

# Utility function to find factorial of n.
def factorial(n) :

    fact = 1;
    for i in range(2, n + 1) :
        fact = fact * i;
    return fact

# Returns count of distinct permutations
# of str.
def countDistinctPermutations(st) :

    length = len(st)
    freq = [0] * MAX_CHAR

    # finding frequency of all the lower
    # case alphabet and storing them in
    # array of integer
    for i in range(0, length) :
        if (st[i] >= 'a') :
            freq[(ord)(st[i]) - 97] = freq[(ord)(st[i]) - 97] + 1;

    # finding factorial of number of
    # appearances and multiplying them
    # since they are repeating alphabets
    fact = 1
    for i in range(0, MAX_CHAR) :
        fact = fact * factorial(freq[i])

    # finding factorial of size of string
    # and dividing it by factorial found
    # after multiplying
    return factorial(length) / fact

# Driver code
st = "fvvfhvgv"
print (countDistinctPermutations(st))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find number of distinct
// permutations of a string.
using System;

public class GFG {

    static int MAX_CHAR = 26;

    // Utility function to find factorial of n.
    static int factorial(int n)
    {
        int fact = 1;
        for (int i = 2; i <= n; i++)
            fact = fact * i;

        return fact;
    }

    // Returns count of distinct permutations
    // of str.
    static int countDistinctPermutations(String str)
    {
        int length = str.Length;

        int[] freq = new int[MAX_CHAR];

        // finding frequency of all the lower case
        // alphabet and storing them in array of
        // integer
        for (int i = 0; i < length; i++)
            if (str[i] >= 'a')
                freq[str[i] - 'a']++;

        // finding factorial of number of appearances
        // and multiplying them since they are
        // repeating alphabets
        int fact = 1;
        for (int i = 0; i < MAX_CHAR; i++)
            fact = fact * factorial(freq[i]);

        // finding factorial of size of string and
        // dividing it by factorial found after
        // multiplying
        return factorial(length) / fact;
    }

    // Driver code
    public static void Main(String []args)
    {
        String str = "fvvfhvgv";

        Console.Write(countDistinctPermutations(str));
    }
}

// This code is contributed by parashar.
```

Output:

```
840
```

Improved By : parashar

Source

https://www.geeksforgeeks.org/number-distinct-permutation-string-can/

Chapter 86

Number of handshakes such that a person shakes hands only once

Number of handshakes such that a person shakes hands only once - GeeksforGeeks

There are N number of persons in a party, find the total number of handshake such that a person can handshake only once.

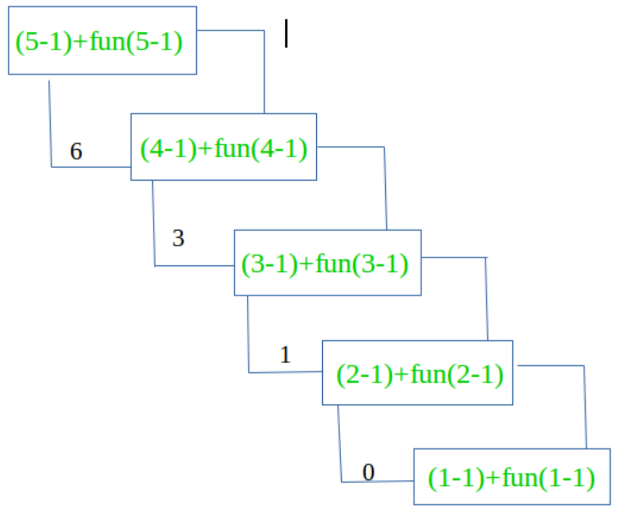
Examples:

```
Input : 5
Output : 10

Input : 9
Output : 36
```

We can see a recursive nature in the problem.

```
// n-th person has (n-1) choices and after
// n-th person chooses a person, problem
// recurs for n-1.
handshake(n) = (n-1) + handshake(n-1)

// Base case
handshake(0) = 0
```

Let assume that number persons are 5.



So here total number of handshake is 10.

Below is implementation of above recursive formula.

C++

```
 // Recursive CPP program to count total
// number of  handshakes when a person
// can shake hand with only one.
#include <stdio.h>

// function to find all possible handshakes
int handshake(int n)
{
    // when n becomes 0 that means all the
    // persons have done handshake with other
    if (n == 0)
        return 0;
    else
        return (n - 1) + handshake(n - 1);
}

int main()
{
    int n = 9;
    printf("%d", handshake(n));
    return 0;
}
```

Java

```
 // Recursive Java program to
// count total number of
// handshakes when a person
// can shake hand with only one.
import java.io.*;

class GFG
{

// function to find all
// possible handshakes
static int handshake(int n)
{
    // when n becomes 0 that
    // means all the persons
    // have done handshake
    // with other
    if (n == 0)
        return 0;
    else
        return (n - 1) + handshake(n - 1);
}

// Driver Code
public static void main (String[] args)
{
    int n = 9;
    System.out.print(handshake(n));
}
}

// This code is contributed
// by chandan_jnu
```

C#

```
 // Recursive C# program to
// count total number of
// handshakes when a person
// can shake hand with only one.
using System;

class GFG
{
```

```
// function to find all
// possible handshakes
static int handshake(int n)
{
    // when n becomes 0 that
    // means all the persons
    // have done handshake
    // with other
    if (n == 0)
        return 0;
    else
        return (n - 1) + handshake(n - 1);
}

// Driver Code
public static void Main (String []args)
{
    int n = 9;
    Console.WriteLine(handshake(n));
}
}

// This code is contributed
// by Arnab Kundu
```

Python3

```
 # Recursive Python program
# to count total number of
# handshakes when a person
# can shake hand with only one.

# function to find all
# possible handshakes
def handshake(n):

    # when n becomes 0 that means
    # all the persons have done
    # handshake with other
    if (n == 0):
        return 0
    else:
        return (n - 1) + handshake(n - 1)

# Driver Code
n = 9
print(handshake(n))
```

```
# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// Recursive PHP program to
// count total number of
// handshakes when a person
// can shake hand with only one.

// function to find all
// possible handshakes
function handshake($n)
{
    // when n becomes 0 that means
    // all the persons have done
    // handshake with other
    if ($n == 0)
        return 0;
    else
        return ($n - 1) + handshake($n - 1);
}

// Driver Code
$n = 9;
echo(handshake($n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
36
```

We can come up with a **direct formula** by expanding the recursion.

```
handshake(n) = (n-1) + handshake(n-1)
             = (n-1) + (n-2) + handshake(n-2)
             = (n-1) + (n-2) + .... 1 + 0
             = n * (n - 1)/2
```

C++

```
 // Recursive CPP program to count total
// number of  handshakes when a person
// can shake hand with only one.
#include <stdio.h>

// function to find all possible handshakes
int handshake(int n)
{
   return n * (n - 1)/2;
}

int main()
{
    int n = 9;
    printf("%d", handshake(n));
    return 0;
}
```

Java

```
 // Recursive Java program to
// count total number of
// handshakes when a person
// can shake hand with only one.
class GFG
{

// function to find all
// possible handshakes
static int handshake(int n)
{
    return n * (n - 1) / 2;
}

// Driver code
public static void main(String args[])
{
    int n = 9;
    System.out.println(handshake(n));
}
}

// This code is contributed
// by Arnab Kundu
```

Python3

```
 # Recursive Python program
```

```
# to count total number of
# handshakes when a person
# can shake hand with only one.

# function to find all
# possible handshakes
def handshake(n):

    return int(n * (n - 1) / 2)

# Driver Code
n = 9
print(handshake(n))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
 <?php
// Recursive PHP program to
// count total number of
// handshakes when a person
// can shake hand with only one.

// function to find all
// possible handshakes
function handshake($n)
{
    return $n * ($n - 1) / 2;
}

// Driver Code
$n = 9;
echo(handshake($n));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
36
```

Improved By : Chandan_Kumar, andrew1234, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/number-of-handshakes-such-that-a-person-shakes-hands-only-once/

Chapter 87

Number of loops of size k starting from a specific node

Number of loops of size k starting from a specific node - GeeksforGeeks

Given two positive integer **n, k**. Consider an undirected complete connected graph of n nodes in a complete connected graph. The task is to calculate the number of ways in which one can start from any node and return to it by visiting K nodes.

Examples:

```
Input : n = 3, k = 3
Output : 2


Input : n = 4, k = 2
Output : 3
```

Lets **f(n, k)** be a function which return number of ways in which one can start from any node and return to it by visiting K nodes.
If we start and end from one node, then we have K – 1 choices to make for the intermediate nodes since we have already chosen one node in the beginning. For each intermediate choice, you have n – 1 options. So, this will yield $(n – 1)^{k – 1}$ but then we have to remove all the choices cause smaller loops, so we subtract **f(n, k − 1)**.
So, recurrence relation becomes,
$f(n, k) = (n – 1)^{k – 1} – f(n, k – 1)$ with base case $f(n, 2) = n – 1$.
On expanding,
$f(n, k) = (n – 1)^{k – 1} – (n – 1)^{k – 2} + (n – 1)^{k – 3}$ ….. (n – 1) (say eqn 1)

Dividing f(n, k) by (n – 1),
$f(n, k)/(n – 1) = (n – 1)^{k – 2} – (n – 1)^{k – 3} + (n – 1)^{k – 4}$ ….. 1 (say eqn 2)

On adding eqn 1 and eqn 2,
f(n, k) + f(n, k)/(n – 1) = (n – 1)$^{k-1}$ + (-1)^{k}
f(n, k) * ((n -1) + 1)/(n – 1) = (n – 1)$^{k-1}$ + (-1)^{k}

[illegible]

Below is the implementation of this approach:

C++

```
 // C++ Program to find number of cycles of length
// k in a graph with n nodes.
#include <bits/stdc++.h>
using namespace std;

// Return the Number of ways from a
// node to make a loop of size K in undirected
// complete connected graph of N nodes
int numOfways(int n, int k)
{
    int p = 1;

    if (k % 2)
        p = -1;

    return (pow(n - 1, k) + p * (n - 1)) / n;
}

// Driven Program
int main()
{
    int n = 4, k = 2;
    cout << numOfways(n, k) << endl;
    return 0;
}
```

Java

```
 // Java Program to find number of
// cycles of length k in a graph
// with n nodes.
public class GFG {

    // Return the Number of ways
    // from a node to make a loop
    // of size K in undirected
    // complete connected graph of
    // N nodes
    static int numOfways(int n, int k)
    {
        int p = 1;

        if (k % 2 != 0)
            p = -1;

        return (int)(Math.pow(n - 1, k)
                    + p * (n - 1)) / n;
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 4, k = 2;

        System.out.println(numOfways(n, k));
    }
}

// This code is contributed by Sam007.
```

Python3

```
 # python Program to find number of
# cycles of length k in a graph
# with n nodes.

# Return the Number of ways from a
# node to make a loop of size K in
# undirected complete connected
# graph of N nodes
def numOfways(n,k):

    p = 1

    if (k % 2):
        p = -1

    return (pow(n - 1, k) +
                   p * (n - 1)) / n

# Driver code
n = 4
k = 2
print (numOfways(n, k))
```

```
# This code is contributed by Sam007.
```

C#

```
 // C# Program to find number of cycles
// of length k in a graph with n nodes.
using System;

class GFG {

    // Return the Number of ways from
    // a node to make a loop of size
    // K in undirected complete
    // connected graph of N nodes
    static int numOfways(int n, int k)
    {
        int p = 1;

        if (k % 2 != 0)
            p = -1;

        return (int)(Math.Pow(n - 1, k)
                     + p * (n - 1)) / n;
    }

    // Driver code
    static void Main()
    {
        int n = 4, k = 2;

        Console.Write( numOfways(n, k) );
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP Program to find number
// of cycles of length
// k in a graph with n nodes.

// Return the Number of ways from a
// node to make a loop of size K
// in undirected complete connected
// graph of N nodes
```

```
function numOfways( $n, $k)
{
$p = 1;

if ($k % 2)
    $p = -1;

return (pow($n - 1, $k) +
        $p * ($n - 1)) / $n;
}

    // Driver Code
    $n = 4;
    $k = 2;
    echo numOfways($n, $k);

// This code is contributed by vt_m.
?>
```

Output:

```
3
```

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/number-ways-node-make-loop-size-k-undirected-complete-connected-graph-n-nodes

Chapter 88

Number of palindromic permutations | Set 1

Number of palindromic permutations | Set 1 - GeeksforGeeks

Given a string str, find count of all palindromic permutations of it.

Examples :

```
Input : str = "gfgf"
Output : 2
There are two palindromic
permutations fggf and gffg

Input : str = "abc"
Output : 0
```

The idea is based on below facts :

- A string can permute to a palindrome if number of odd occurring characters are at most one.
- One occurrence of the only odd character always goes to middle.
- Half of counts of all characters decide the result. In case of odd occurring character it is floor of half. Other half is automatically decided

For example if input string is “aabbccd”, the count of palindromic permutations is 3! (We get three by taking floor of half of all counts)

What if half itself has repeated characters?
We apply simple combinatorial rule and divide by factorial of half.

For example “aaaaaabbbb”, floor of half of string is 5. In half of a palindromic string, ‘a’ is repeated three times and ‘b’ is repeated two times, so our result is (5!) / (2!) * (3!).

C++

```
 // CPP program to find number of
// palindromic permutations of a
// given string
#include <bits/stdc++.h>
using namespace std;

const int MAX = 256;

// Returns factorial of n
long long int fact(int n)
{
    long long int res = 1;
    for (int i = 2; i <= n; i++)
        res = res * i;
    return res;
}

// Returns count of palindromic
// permutations of str.
int countPalinPermutations(string &str)
{
    // Count frequencies of all characters
    int n = str.length();
    int freq[MAX] = { 0 };
    for (int i = 0; i < n; i++)
        freq[str[i]]++;

    // Since half of the characters
    // decide count of palindromic
    // permutations, we take (n/2)!
    long long int res = fact(n / 2);

    // To make sure that there is at
    // most one odd occurring char
    bool oddFreq = false;

    // Traverse through all counts
    for (int i = 0; i < MAX; i++) {
        int half = freq[i] / 2;

        // To make sure that the
        // string can permute to
        // form a palindrome
        if (freq[i] % 2 != 0) {

            // If there are more than
            // one odd occurring chars
            if (oddFreq == true)
                return 0;
            oddFreq = true;
        }

        // Divide all permutations with
        // repeated characters
        res = res / fact(half);
    }

    return res;
}

// Driver code
int main()
{
    string str = "gffg";
    cout << countPalinPermutations(str);
    return 0;
}
```

Java

```
 // Java program to find number of
// palindromic permutations of a
// given string
class GFG {

    static final int MAX = 256;

    // Returns factorial of n
    static long fact(int n)
    {
        long res = 1;

        for (int i = 2; i <= n; i++)
            res = res * i;

        return res;
    }

    // Returns count of palindromic
    // permutations of str.
    static int countPalinPermutations(String str)
    {

        // Count frequencies of all characters
        int n = str.length();
        int freq[]=new int[MAX];

        for (int i = 0; i < n; i++)
            freq[str.charAt(i)]++;

        // Since half of the characters
        // decide count of palindromic
        // permutations, we take (n/2)!
        long res = fact(n / 2);

        // To make sure that there is at
        // most one odd occurring char
        boolean oddFreq = false;

        // Traverse through all counts
        for (int i = 0; i < MAX; i++) {
            int half = freq[i] / 2;

            // To make sure that the
            // string can permute to
            // form a palindrome
            if (freq[i] % 2 != 0) {

                // If there are more than
                // one odd occurring chars
                if (oddFreq == true)
                    return 0;

                oddFreq = true;
            }

            // Divide all permutations with
            // repeated characters
            res = res / fact(half);
        }

        return (int)res;
    }

    // Driver code
    public static void main (String[] args)
    {

        String str = "gffg";

        System.out.print(
            countPalinPermutations(str));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find number of
# palindromic permutations of a
# given string
MAX = 256

# Returns factorial of n
def fact(n) :
    res = 1
    for i in range(2, n+1) :
        res = res * i
    return res

# Returns count of palindromic
# permutations of str.
def countPalinPermutations(str) :
    global MAX

    # Count frequencies of
    # all characters
    n = len(str)
    freq = [0] * MAX;
    for i in range(0, n) :
        freq[ord(str[i])] = freq[ord(str[i])] + 1;
    # Since half of the characters
    # decide count of palindromic
    # permutations, we take (n/2)!
    res = fact(int(n / 2))

    # To make sure that there is at
    # most one odd occurring char
    oddFreq = False

    # Traverse through all counts
    for i in range(0, MAX) :
        half = int(freq[i] / 2)

        # To make sure that the
        # string can permute to
        # form a palindrome
        if (freq[i] % 2 != 0):
            # If there are more than
            # one odd occurring chars
            if (oddFreq == True):
                return 0
            oddFreq = True

        # Divide all permutations
        # with repeated characters
        res = int(res / fact(half))

    return res

# Driver code
str = "gffg"
print (countPalinPermutations(str))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // C# program to find number of
// palindromic permutations of a
// given string
using System;

class GFG {

    static int MAX = 256;

    // Returns factorial of n
    static long fact(int n)
    {
        long res = 1;

        for (int i = 2; i <= n; i++)
            res = res * i;

        return res;
    }

    // Returns count of palindromic
    // permutations of str.
    static int countPalinPermutations(string str)
    {

        // Count frequencies of all characters
        int n = str.Length;
        int []freq=new int[MAX];

        for (int i = 0; i < n; i++)
            freq[str[i]]++;

        // Since half of the characters
        // decide count of palindromic
        // permutations, we take (n/2)!
        long res = fact(n / 2);

        // To make sure that there is at
        // most one odd occurring char
        bool oddFreq = false;

        // Traverse through all counts
        for (int i = 0; i < MAX; i++) {
            int half = freq[i] / 2;

            // To make sure that the
            // string can permute to
            // form a palindrome
            if (freq[i] % 2 != 0) {

                // If there are more than
                // one odd occurring chars
                if (oddFreq == true)
                    return 0;

                oddFreq = true;
            }

            // Divide all permutations with
            // repeated characters
            res = res / fact(half);
        }

        return (int)res;
    }

    // Driver code
    public static void Main ()
    {

        string str = "gffg";

        Console.WriteLine(
            countPalinPermutations(str));
    }
```

```
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number of
// palindromic permutations of a
// given string
$MAX = 256;

// Returns factorial of n
function fact($n)
{
    $res = 1;
    for ($i = 2; $i <= $n; $i++)
        $res = $res * $i;
    return $res;
}

// Returns count of palindromic
// permutations of str.
function countPalinPermutations(&$str)
{
    global $MAX ;

    // Count frequencies of
    // all characters
    $n = strlen($str);
    $freq = (0);
    for ($i = 0; $i < $n; $i++)

    // Since half of the characters
    // decide count of palindromic
    // permutations, we take (n/2)!
    $res = fact($n / 2);

    // To make sure that there is at
    // most one odd occurring char
    $oddFreq = false;

    // Traverse through all counts
    for ($i = 0; $i < $MAX; $i++)
    {
        $half = $freq[$i] / 2;

        // To make sure that the
        // string can permute to
        // form a palindrome
        if ($freq[$i] % 2 != 0)
        {

            // If there are more than
            // one odd occurring chars
            if ($oddFreq == true)
                return 0;
            $oddFreq = true;
        }

        // Divide all permutations
        // with repeated characters
        $res = $res / fact($half);
    }

    return $res;
}

// Driver code
$str = "gffg";
echo countPalinPermutations($str);

// This code is contributed by ajit
?>
```

Output :

```
2
```

The above solution causes overflow very early. We can avoid overflow by doing modular arithmetic. In the next article, we would be discussing modular arithmetic based approach.

Improved By : JainHarshit, jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/number-of-palindromic-permutations-set-1/

Chapter 89

Number of permutations such that sum of elements at odd index and even index are equal

Number of permutations such that sum of elements at odd index and even index are equal - GeeksforGeeks

Given N numbers, find the number of permutations in which the sum of elements at odd index and sum of elements at even index are equal.

Examples:

> **Input:** 1 2 3
> **Output:** 2
> The permutations are:
> 1 3 2 sum at odd index = 1+2 = 3, sum at even index = 3
> 2 3 1 sum at odd index = 2+1 = 3, sum at even index = 3
>
> **Input:** 1 2 1 2
> **Output:** 3
> The permutations are:
> 1 2 2 1
> 2 1 1 2
> 2 2 1 1

The **approach** to the problem will be to use next_permutation() in C++ STL which helps to generate all the possible permutation of N numbers. If the sum of the odd index elements is equal to the sum of even index elements of the generated permutation, then increase the count. When all permutations are checked, print the count.

Below is the implementation of the above approach:

```
 // C++ program to find number of permutations
// such that sum of elements at odd index
// and even index are equal
#include <bits/stdc++.h>
using namespace std;

// Function that returns the number of permutations
int numberOfPermutations(int a[], int n)
{
    int sumEven, sumOdd, c = 0;

    // iterate for all permutations
    do {
        // stores the sum of odd and even index elements
        sumEven = sumOdd = 0;

        // iterate for elements in permutatio
        for (int i = 0; i < n; i++) {

            // if odd index
            if (i % 2)
                sumOdd += a[i];
            else
                sumEven += a[i];
        }

        // If condition holds
        if (sumOdd == sumEven)
            c++;

    } while (next_permutation(a, a + n));

    // return the number of permutations
    return c;
}
// Driver Code
int main()
{

    int a[] = { 1, 2, 3 };
    int n = sizeof(a) / sizeof(a[0]);

    // Calling Function
    cout << numberOfPermutations(a, n);

    return 0;
}
```

Output:

```
2
```

Time Complexity: O(N! * N)

Source

https://www.geeksforgeeks.org/number-of-permutations-such-that-sum-of-elements-at-odd-index-and-even-index-a

Chapter 90

Number of strings of length N with no palindromic sub string

Number of strings of length N with no palindromic sub string - GeeksforGeeks

Given two positive integers **N, M**. The task is to find the number of strings of length N under the alphabet set of size M such that no substrings of size greater than 1 is palindromic.

Examples:

```
Input : N = 2, M = 3
Output : 6
In this case, set of alphabet are 3, say {A, B, C}
All possible string of length 2, using 3 letters are:
{AA, AB, AC, BA, BB, BC, CA, CB, CC}
Out of these {AA, BB, CC} contain palindromic substring,
so our answer will be
8 - 2 = 6.

Input : N = 2, M = 2
Output : 2
Out of {AA, BB, AB, BA}, only {AB, BA} contain
non-palindromic substrings.
```

First, observe, a string does not contain any palindromic substring if the string doesn't have any palindromic substring of the length 2 and 3, because all the palindromic string of the greater lengths contains at least one palindromic substring of the length of 2 or 3, basically in the center.

So, the following is true:

- There are M ways to choose the first symbol of the string.

- Then there are (M – 1) ways to choose the second symbol of the string. Basically, it should not be equal to first one.
- Then there are (M – 2) ways to choose any next symbol. Basically, it should not coincide with the previous symbols, that aren't equal.

Knowing this, we can evaluate the answer in the following ways:

- If N = 1, then the answer will be M.
- If N = 2, then the answer is M*(M – 1).
- If N >= 3, then M * (M – 1) * (M – 2)$^{N-2}$.

Below is the implementation of above idea :

C++

```
 // CPP program to count number of strings of
// size m such that no substring is palindrome.
#include <bits/stdc++.h>
using namespace std;

// Return the count of strings with
// no palindromic substring.
int numofstring(int n, int m)
{
    if (n == 1)
        return m;

    if (n == 2)
        return m * (m - 1);

    return m * (m - 1) * pow(m - 2, n - 2);
}

// Driven Program
int main()
{
    int n = 2, m = 3;
    cout << numofstring(n, m) << endl;
    return 0;
}
```

Java

```
 // Java program to count number of strings of
// size m such that no substring is palindrome.
import java.io.*;
```

```
class GFG {

    // Return the count of strings with
    // no palindromic substring.
    static int numofstring(int n, int m)
    {
        if (n == 1)
            return m;

        if (n == 2)
            return m * (m - 1);

        return m * (m - 1) * (int)Math.pow(m - 2, n - 2);
    }

    // Driven Program
    public static void main (String[] args)
    {
        int n = 2, m = 3;
        System.out.println(numofstring(n, m));
    }
}

// This code is contributed by ajit.
```

Python3

```
 # Python3 program to count number of strings of
# size m such that no substring is palindrome

# Return the count of strings with
# no palindromic substring.
def numofstring(n, m):
    if n == 1:
        return m

    if n == 2:
        return m * (m - 1)

    return m * (m - 1) * pow(m - 2, n - 2)

# Driven Program
n = 2
m = 3
print (numofstring(n, m))

# This code is contributed
# by Shreyanshi Arun.
```

C#

```
 // C# program to count number of strings of
// size m such that no substring is palindrome.
using System;

class GFG {

    // Return the count of strings with
    // no palindromic substring.
    static int numofstring(int n, int m)
    {
        if (n == 1)
            return m;

        if (n == 2)
            return m * (m - 1);

        return m * (m - 1) * (int)Math.Pow(m - 2,
                                           n - 2);
    }

    // Driver Code
    public static void Main ()
    {
        int n = 2, m = 3;
        Console.Write(numofstring(n, m));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to count number
// of strings of size m such
// that no substring is palindrome.

// Return the count of strings with
// no palindromic substring.
function numofstring($n, $m)
{
    if ($n == 1)
        return $m;
    if ($n == 2)
        return $m * ($m - 1);

    return $m * ($m - 1) *
           pow($m - 2, $n - 2);
}

// Driver Code
{
    $n = 2; $m = 3;
    echo numofstring($n, $m) ;
    return 0;
}

// This code is contributed by nitin mittal.
?>
```

Output

```
6
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/number-string-length-n-no-palindromic-sub-string/

Chapter 91

Number of subsequences as “ab” in a string repeated K times

Number of subsequences as ”ab” in a string repeated K times - GeeksforGeeks

Given a String S, consider a new string formed by repeating the S exactly K times. We need find the number of subsequences as “ab” in the newly formed string.

Examples :

```
Input : S = "abcb"
        K = 2
Output : 6
Here, Given string is repeated 2 times and
we get a new string "abcbabcb"
Below are 6 occurrences of "ab"
abcbabcb
abcbabcb
abcbabcb
abcbabcb
abcbabcb

Input : S = "aacbd"
        K = 1
Output : 2
```

Naive Approach: Finding no.of subsequences of “ab” is in fact finding a pair s[i], s[j] (i < j) where s[i] = ‘a’, s[j] = ‘b’.
We can do this by using two nested for loops and count the no. of pairs.

We can improve this approach in a single traversal of the string. Let us consider an index j, **s[j] ='b'**, if we find no.of index i's such that **s[i] = 'a'** and i < j, then it is same as no.of subsequences of "ab" till j. This can be done by maintaining count of a's by traversing the array and add the count to our answer at position where s[i] ='b .
Time Complexity:O(|S|*K)

Efficient Approach:
Let T be the newly formed string
T = s1 + s2 + s3 + ….. + sk;
where si is the ith occurrence of the string s.
Here, occurrence of "ab" in T are as follows:
1)"ab" lies completely in the some of occurrence of string S, so we can simply find occurrences of "ab" in Si.Let it be C. So, total no.of occurrences of "ab" in T will be **C*K**.

2) Otherwise, "a" lies strictly inside some string Si and "b" lies inside some other string Sj, (i < j). In this way finding no.of occurrences of "ab" will be choosing two occurrences of string S from K occurrences$(_{\mathbf{KC_2}})$ and multiplying it with no. of occurrences of "a" in Si and no.of occurrences of "b" in Sj.
As, Si = Sj = S.
Time Complexity: O(|S|), for counting no.of "a"s and no.of "b"s.

C++

```
 // CPP code to find number of subsequences of
// "ab" in the string S which is repeated K times.
#include <bits/stdc++.h>
using namespace std;

int countOccurrences(string s, int K)
{
    int n = s.length();
    int C, c1 = 0, c2 = 0;
    for (int i = 0; i < n; i++) {
        if (s[i] == 'a')
            c1++; // Count of 'a's
        if (s[i] == 'b') {
            c2++; // Count of 'b's
            C += c1; // occurrence of "ab"s in string S
        }
    }

    // Add following two :
    // 1) K * (Occurrences of "ab" in single string)
    // 2) a is from one string and b is from other.
    return C * K + (K * (K - 1) / 2) * c1 * c2;
}

// Driver code
```

```
int main()
{
    string S = "abcb";
    int k = 2;
    cout << countOccurrences(S, k) << endl;
    return 0;
}
```

Java

```
 // Java code to find number of subsequences of
// "ab" in the string S which is repeated K times.

import java.io.*;

class GFG {

    static int countOccurrences(String s, int K)
    {
        int n = s.length();
        int C = 0, c1 = 0, c2 = 0;
        for (int i = 0; i < n; i++) {
            if (s.charAt(i) == 'a')
                c1++; // Count of 'a's
            if (s.charAt(i) == 'b') {
                c2++; // Count of 'b's

                // occurrence of "ab"s
                // in string S
                C += c1;
            }
        }

        // Add following two :
        // 1) K * (Occurrences of "ab" in single string)
        // 2) a is from one string and b is from other.
        return C * K + (K * (K - 1) / 2) * c1 * c2;
    }

    // Driver code
    public static void main(String[] args)
    {
        String S = "abcb";
        int k = 2;

        System.out.println(countOccurrences(S, k));
    }
}
```

```
// This code is contributed by vt_m.
```

Python3

```
 # Python3 code to find number of
# subsequences of "ab" in the
# string S which is repeated K times.

def countOccurrences (s, K):
    n = len(s)
    c1 = 0
    c2 = 0
    C = 0
    for i in range(n):
        if s[i] == 'a':
            c1+= 1 # Count of 'a's
        if s[i] == 'b':
            c2+= 1 # Count of 'b's

            # occurrence of "ab"s in string S
            C += c1

    # Add following two :
    # 1) K * (Occurrences of "ab" in single string)
    # 2) a is from one string and b is from other.
    return C * K + (K * (K - 1) / 2) * c1 * c2


# Driver code
S = "abcb"
k = 2
print (countOccurrences(S, k))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# code to find number of subsequences
// of "ab" in the string S which is
// repeated K times.
using System;

class GFG {

    static int countOccurrences(string s, int K)
    {
        int n = s.Length;
        int C = 0, c1 = 0, c2 = 0;

        for (int i = 0; i < n; i++) {

            if (s[i] == 'a')

                // Count of 'a's
                c1++;
            if (s[i] == 'b') {

                // Count of 'b's
                c2++;

                // occurrence of "ab"s
                // in string S
                C += c1;
            }
        }

        // Add following two :
        // 1) K * (Occurrences of "ab" in
        // single string)
        // 2) a is from one string and b
        // is from other.
        return C * K + (K * (K - 1) / 2)
                                * c1 * c2;
    }

    // Driver code
    public static void Main()
    {

        string S = "abcb";
        int k = 2;

        Console.WriteLine(
                  countOccurrences(S, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to find number of
// subsequences of "ab" in the
// string S which is repeated K times.

function countOccurrences($s, $K)
{
    $n = strlen($s);
    $C = 0; $c1 = 0; $c2 = 0;
    for ($i = 0; $i < $n; $i++)
    {
        if ($s[$i] == 'a')
            // Count of 'a's
            $c1++;
        if ($s[$i] == 'b')
        {
            // Count of 'b's
            $c2++;

            // occurrence of "ab"s
            // in string S
            $C = $C+ $c1;
        }
    }

    // Add following two :
    // 1) K * (Occurrences of "ab"
    //    in single string)
    // 2) a is from one string and
    //    b is from other.
    return $C * $K + ($K * ($K - 1) / 2) *
                                $c1 * $c2;
}

// Driver code
$S = "abcb";
$k = 2;
echo countOccurrences($S, $k) ,"\n";

// This code is contributed by ajit.
?>
```

Output :

```
6
```

Time complexity:O(|S|).

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/number-subsequences-ab-string-repeated-k-times/

Chapter 92

Number of ways to cut a stick of length N into K pieces

Number of ways to cut a stick of length N into K pieces - GeeksforGeeks

Given a stick of size N, find the number of ways in which it can be cut into K pieces such that length of every piece is greater than 0.

Examples :

```
Input : N = 5
        K = 2
Output : 4


Input : N = 15
        K = 5
Output : 1001
```

Solving this question is equivalent to solving the mathematics equation $\mathbf{x_1 + x_2 + ... + x_K = N}$
We can solve this by using the **bars and stars** method in Combinatorics, from which we obtain the fact that the number of positive integral solutions to this equation is $^{\mathbf{(N-1)}}\mathbf{C}_{\mathbf{(K-1)}}$, where $^{N}C_{K}$ is N! / ((N – K) ! * (K!)), where ! stands for factorial.

In C++ and Java, for large values of factorials, there might be overflow errors. In that case we can introduce a large prime number such as $10^7 + 7$ to mod the answer. We can calculate nCr % p by using Lucas Theorem.
However, python can handle large values without overflow.

C++

```
 // C++ program to calculate the number of ways
```

```
// to divide a stick of length n into k pieces
#include <bits/stdc++.h>
using namespace std;

// function to generate nCk or nChoosek
unsigned long long nCr(unsigned long long n,
                       unsigned long long r)
{
    if (n < r)
        return 0;

    // Reduces to the form n! / n!
    if (r == 0)
        return 1;

    // nCr has been simplified to this form by
    // expanding numerator and denominator to
    // the form   n(n - 1)(n - 2)...(n - r + 1)
    //             -----------------------------
    //                         (r!)
    // in the above equation, (n - r)! is cancelled
    // out in the numerator and denominator

    unsigned long long numerator = 1;
    for (int i = n; i > n - r; i--)
        numerator = (numerator * i);

    unsigned long long denominator = 1;
    for (int i = 1; i < r + 1; i++)
        denominator = (denominator * i);

    return (numerator / denominator);
}

// Returns number of ways to cut
// a rod of length N into K peices.
unsigned long long countWays(unsigned long long N,
                             unsigned long long K)
{
    return nCr(N - 1, K - 1);
}

// Driver code
int main()
{
    unsigned long long N = 5;
    unsigned long long K = 2;
    cout << countWays(N, K);
    return 0;
}
```

Java

```
 // Java program to find the number of ways in which
// a stick of length n can be divided into K pieces
import java.io.*;
import java.util.*;

class GFG
{
    // function to generate nCk or nChoosek
    public static int nCr(int n, int r)
    {
        if (n < r)
            return 0;

        // Reduces to the form n! / n!
        if (r == 0)
            return 1;

        // nCr has been simplified to this form by
        // expanding numerator and denominator to
        // the form  n(n - 1)(n - 2)...(n - r + 1)
        //             -----------------------------
        //                          (r!)
        // in the above equation, (n-r)! is cancelled
        // out in the numerator and denominator

        int numerator = 1;
        for (int i = n ; i > n - r ; i--)
            numerator = (numerator * i);

        int denominator = 1;
        for (int i = 1 ; i < r + 1 ; i++)
            denominator = (denominator * i);

        return (numerator / denominator);
    }

    // Returns number of ways to cut
    // a rod of length N into K peices
    public static int countWays(int N, int K)
    {
        return nCr(N - 1, K - 1);
    }
    public static void main(String[] args)
    {
        int N = 5;
        int K = 2;
        System.out.println(countWays(N, K));
    }
}
```

Python3

```
 # Python program to find the number
# of ways  in which a stick of length
# n can be divided into K pieces

# function to generate nCk or nChoosek
def nCr(n, r):

    if (n < r):
        return 0

    # reduces to the form n! / n!
    if (r == 0):
        return 1

    # nCr has been simplified to this form by
    # expanding numerator and denominator to
    # the form     n(n - 1)(n - 2)...(n - r + 1)
    #             -----------------------------
    #                        (r!)
    # in the above equation, (n-r)! is cancelled
    # out in the numerator and denominator

    numerator = 1
    for i in range(n, n - r, -1):
        numerator = numerator * i

    denominator = 1
    for i in range(1, r + 1):
        denominator = denominator * i

    return (numerator // denominator)

# Returns number of ways to cut
# a rod of length N into K peices.
def countWays(N, K) :
    return nCr(N - 1, K - 1);

# Driver code
```

```
N = 5
K = 2
print(countWays(N, K))
```

C#

```
 // C# program to find the number of
// ways in which a stick of length n
// can be divided into K pieces
using System;

class GFG
{
    // function to generate nCk or nChoosek
    public static int nCr(int n, int r)
    {
        if (n < r)
            return 0;

        // Reduces to the form n! / n!
        if (r == 0)
            return 1;

        // nCr has been simplified to this form by
        // expanding numerator and denominator to
        // the form  n(n - 1)(n - 2)...(n - r + 1)
        //              -----------------------------
        //                           (r!)
        // in the above equation, (n-r)! is cancelled
        // out in the numerator and denominator

        int numerator = 1;
        for (int i = n; i > n - r; i--)
            numerator = (numerator * i);

        int denominator = 1;
        for (int i = 1; i < r + 1; i++)
            denominator = (denominator * i);

        return (numerator / denominator);
    }

    // Returns number of ways to cut
    // a rod of length N into K pieces
    public static int countWays(int N, int K)
    {
        return nCr(N - 1, K - 1);
    }
    public static void Main()
    {
        int N = 5;
        int K = 2;
        Console.Write(countWays(N, K));

    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to calculate the
// number of ways to divide a
// stick of length n into k pieces


// function to generate nCk or nChoosek
function nCr($n, $r)
{
    if ($n < $r)
        return 0;

    // Reduces to the form n! / n!
    if ($r == 0)
        return 1;

    // nCr has been simplified to this form by
    // expanding numerator and denominator to
    // the form n(n - 1)(n - 2)...(n - r + 1)
    //             -----------------------------
    //                         (r!)
    // in the above equation, (n - r)! is cancelled
    // out in the numerator and denominator

    $numerator = 1;
    for ($i = $n; $i > $n - $r; $i--)
        $numerator = ($numerator * $i);

    $denominator = 1;
    for ($i = 1; $i < $r + 1; $i++)
        $denominator = ($denominator * $i);

    return (floor($numerator / $denominator));
}
```

```
// Returns number of ways to cut
// a rod of length N into K peices.
function countWays($N, $K)
{
    return nCr($N - 1, $K - 1);
}

// Driver code
$N = 5;
$K = 2;
echo countWays($N, $K);
return 0;

// This code is contributed by nitin mittal.
?>
```

Output :

```
4
```

Exercise :
Extend the above problem with 0 length pieces allowed. Hint : The number of solutions can similarly be found by writing each x_i as $y_i - 1$, and we get an equation $\mathbf{y_1 + y_2 + \ldots.. + y_K = N + K}$. The number of solutions to this equation is $\mathbf{{}^{(N + K - 1)}C_{(K - 1)}}$

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/number-ways-cut-stick-length-n-k-pieces/

Chapter 93

Number of ways to form a heap with n distinct integers

Number of ways to form a heap with n distinct integers - GeeksforGeeks

Given n, how many distinctMax Heapcan be made from n distinct integers?

Examples:

```
Input : n = 3
Output : Assume the integers are 1, 2, 3.
Then the 2 possible max heaps are:
            3
           / \
          1   2

           3
          / \
         2   1

Input : n = 4
Output : Assume the integers are 1, 2, 3, 4.
Then the 3 possible max heaps are:
        4
       / \
      3   2
     /
    1

        4
       / \
      2   3
```

```
  /
 1

     4
    / \
   3   1
  /
 2
```

Since there is only one element as the **root**, it must be the largest number. Now we have n-1 remaining elements. The main observation here is that because of the max heap properties, the **structure** of the heap nodes will remain the same in all instances, but only the values in the nodes will change.

Assume there are **l** elements in the **left sub-tree** and **r** elements in the **right sub-tree**. Now for the root, l + r = n-1. From this we can see that we can **choose** any l of the remaining n-1 elements for the left sub-tree as they are all smaller than the root.

We know the there are $\binom{n-1}{l}$ ways to do this. Next for each instance of these, we can have many heaps with l elements and for each of those we can have many heaps with r elements. Thus we can consider them as subproblems and **recur** for the final answer as:

T(n) = $\binom{n-1}{l}$ * T(L) * T(R).

Now we have to find the **values** for l and r for a given n. We know that the height of the heap h = $\log_2 n$. Also the maximum number of elements that can be present in the h th **level** of any heap, m = 2^h, where the root is at the 0th level. Moreover the number of elements actually present in the last level of the heap p = n – (2^h – 1). (since $2^h - 1$ number of nodes present till the penultimate level). Thus, there can be two **cases**: when the last level is more than or equal to half-filled:

l = 2^h – 1, if p >= m / 2
(or) the last level is less than half-filled:

l = 2^h – 1 – ((m / 2) – p), if p < m / 2

(we get 2^h – 1 here because left subtree has $2^0 + 2^1 + .. + 2^{h-2}$ nodes.
From this we can also say that r = n – l – 1.

We can use the **dynamic programming** approach discussed in this post here to find the values of $\binom{n}{k}$. Similarly if we look at the recursion tree for the **optimal substructure** recurrence formed above, we can see that it also has **overlapping subproblems** property, hence can be solved using dynamic programming:

```
             T(7)
```

```
            /    \
          T(3)   T(3)
         /  \     /  \
     T(1)  T(1) T(1) T(1)
```

Following is the c++ implementation of the above approach:

```
 // CPP program to count max heaps with n distinct keys
#include <iostream>
using namespace std;

#define MAXN 105 // maximum value of n here

// dp[i] = number of max heaps for i distinct integers
int dp[MAXN];

// nck[i][j] = number of ways to choose j elements
//             form i elements, no order */
int nck[MAXN][MAXN];

// log2[i] = floor of logarithm of base 2 of i
int log2[MAXN];

// to calculate nCk
int choose(int n, int k)
{
    if (k > n)
        return 0;
    if (n <= 1)
        return 1;
    if (k == 0)
        return 1;

    if (nck[n][k] != -1)
        return nck[n][k];

    int answer = choose(n - 1, k - 1) + choose(n - 1, k);
    nck[n][k] = answer;
    return answer;
}

// calculate l for give value of n
int getLeft(int n)
{
    if (n == 1)
        return 0;

    int h = log2[n];
    // max number of elements that can be present in the
    // hth level of any heap
    int numh = (1 << h); //(2 ^ h)

    // number of elements that are actually present in
    // last level(hth level)
    // (2^h - 1)
    int last = n - ((1 << h) - 1);

    // if more than half-filled
    if (last >= (numh / 2))
        return (1 << h) - 1; // (2^h) - 1
    else
        return (1 << h) - 1 - ((numh / 2) - last);
}

// find maximum number of heaps for n
int numberOfHeaps(int n)
{
    if (n <= 1)
        return 1;

    if (dp[n] != -1)
        return dp[n];

    int left = getLeft(n);
    int ans = (choose(n - 1, left) * numberOfHeaps(left)) *
                              (numberOfHeaps(n - 1 - left));
    dp[n] = ans;
    return ans;
}

// function to initialize arrays
int solve(int n)
{
    for (int i = 0; i <= n; i++)
        dp[i] = -1;

    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++)
            nck[i][j] = -1;

    int currLog2 = -1;
    int currPower2 = 1;

    // for each power of two find logarithm
    for (int i = 1; i <= n; i++) {
        if (currPower2 == i) {
            currLog2++;
            currPower2 *= 2;
        }
        log2[i] = currLog2;
    }

    return numberOfHeaps(n);
}

// driver function
int main()
{
    int n = 10;
    cout << solve(n) << endl;
    return 0;
}
```

Output:

```
3360
```

Asked in: Directi.

Improved By : rsatish1110

Source

https://www.geeksforgeeks.org/number-ways-form-heap-n-distinct-integers/

Chapter 94

Number of ways to form an array with distinct adjacent elements

Number of ways to form an array with distinct adjacent elements - GeeksforGeeks

Given three integers N, M and X, the task is to find the number of ways to form an array, such that all consecutive numbers of the array are distinct, and the value at any index of the array from 2 to N – 1(Considering 1 based indexing) lies between 1 and M, while the value at index 1 is X and the value at index N is 1.
Note: Value of X lies between 1 and M.

Examples:

> **Input:** N = 4, M = 3, X = 2
> **Output:** 3
> The following arrays are possible:
> 1) 2, 1, 2, 1
> 2) 2, 1, 3, 1
> 3) 2, 3, 2, 1
>
> **Input:** N = 2, M = 3, X = 2
> **Output:** 1
> The only possible array is: 2, 1

Approach: The problem can be solved using Dynamic Programming. Let, f(i) represent the number of ways to form the array till the ith index, such that every consecutive element of the array is distinct. Let f(i, One) represent the number of ways to form the array till the i-th index such that every consecutive element of the array is distinct and $arr_i = 1$.
Similarly, let **f(i, Non-One)** represent the number of ways to form the array till the ith index, such that every consecutive element of the array is distinct and arr_i is not equal to 1.

The following recurrence is formed:

```
f(i, Non-One) = f(i - 1, One) * (M - 1) + f(i - 1, Non-One) * (M - 2)
```

which means that the number of ways to form the array till the ith index with array_{i} not equal to 1 is formed using two cases:

1. If the number at $\text{array}_{\text{i − 1}}$ is 1, then opt one number out of **(M − 1)** options to place at the i^{th} index, since array_{i} is not equal to 1.
2. If the number at $\text{array}_{\text{i − 1}}$ is not 1, then we need to opt one number out of **(M − 2)** options, since array_{i} is not equal to 1 and array_{i} $\text{array}_{\text{i − 1}}$.

Similarly, **f(i, One) = f(i − 1, Non-One)**, since the number of ways to form the array till the ith index with array_{i} = 1, is same as number of ways to form the array till the (i – 1)th index with $\text{array}_{\text{i − 1}}$ 1, thus at the i^{th} index there is only one option. At the end the required answer if f(N, One) since array_{N} needs to be equal to 1.

Below is the implementation of the above approach:

C++

```
 // C++ program to count the number of ways
// to form arrays of N numbers such that the
// first and last numbers are fixed
// and all consecutive numbers are distinct
#include <bits/stdc++.h>
using namespace std;

// Returns the total ways to form arrays such that
// every consecutive element is different and each
// element except the first and last can take values
// from 1 to M
int totalWays(int N, int M, int X)
{

    // define the dp[][] array
    int dp[N + 1][2];

    // if the first element is 1
    if (X == 1) {

        // there is only one way to place
        // a 1 at the first index
        dp[0][0] = 1;
    }
    else {

        // the value at first index needs to be 1,
        // thus there is no way to place a non-one integer
        dp[0][1] = 0;
    }

    // if the first element was 1 then at index 1,
    // only non one integer can be placed thus
    // there are M - 1 ways to place a non one integer
    // at index 2 and 0 ways to place a 1 at the 2nd index
    if (X == 1) {
        dp[1][0] = 0;
        dp[1][1] = M - 1;
    }

    // Else there is one way to place a one at
    // index 2 and if a non one needs to be placed here,
    // there are (M - 2) options, i.e
    // neither the element at this index
    // should be 1, neither should it be
    // equal to the previous element
    else {
        dp[1][0] = 1;
        dp[1][1] = (M - 2);
    }

    // Build the dp array in bottom up manner
    for (int i = 2; i < N; i++) {

        // f(i, one) = f(i - 1, non-one)
        dp[i][0] = dp[i - 1][1];

        // f(i, non-one) = f(i - 1, one) * (M - 1) +
        // f(i - 1, non-one) * (M - 2)
        dp[i][1] = dp[i - 1][0] * (M - 1) + dp[i - 1][1] * (M - 2);
    }

    // last element needs to be one, so return dp[n - 1][0]
    return dp[N - 1][0];
}

// Driver Code
int main()
{

    int N = 4, M = 3, X = 2;
    cout << totalWays(N, M, X) << endl;

    return 0;
}
```

Java

```
 // Java program to count the
// number of ways to form
// arrays of N numbers such
// that the first and last
// numbers are fixed and all
// consecutive numbers are
// distinct
import java.io.*;

class GFG
{

// Returns the total ways to
// form arrays such that every
// consecutive element is
// different and each element
// except the first and last
// can take values from 1 to M
static int totalWays(int N,
                     int M, int X)
{

    // define the dp[][] array
    int dp[][] = new int[N + 1][2];

    // if the first element is 1
    if (X == 1)
    {

        // there is only one
        // way to place a 1
        // at the first index
        dp[0][0] = 1;
    }
    else
    {

        // the value at first index
        // needs to be 1, thus there
        // is no way to place a
        // non-one integer
        dp[0][1] = 0;
    }

    // if the first element was 1
    // then at index 1, only non
    // one integer can be placed
    // thus there are M - 1 ways
    // to place a non one integer
    // at index 2 and 0 ways to
    // place a 1 at the 2nd index
    if (X == 1)
    {
        dp[1][0] = 0;
        dp[1][1] = M - 1;
    }

    // Else there is one way to
    // place a one at index 2
    // and if a non one needs to
    // be placed here, there are
    // (M - 2) options, i.e neither
    // the element at this index
    // should be 1, neither should
    // it be equal to the previous
    // element
    else
    {
        dp[1][0] = 1;
        dp[1][1] = (M - 2);
    }

    // Build the dp array
    // in bottom up manner
    for (int i = 2; i < N; i++)
    {

        // f(i, one) = f(i - 1,
        // non-one)
        dp[i][0] = dp[i - 1][1];

        // f(i, non-one) =
        // f(i - 1, one) * (M - 1) +
        // f(i - 1, non-one) * (M - 2)
        dp[i][1] = dp[i - 1][0] * (M - 1) +
                   dp[i - 1][1] * (M - 2);
    }

    // last element needs to be
    // one, so return dp[n - 1][0]
    return dp[N - 1][0];
}

// Driver Code
```

```
public static void main (String[] args)
{
    int N = 4, M = 3, X = 2;
    System.out.println(totalWays(N, M, X));
}
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to count the
// number of ways to form
// arrays of N numbers such
// that the first and last
// numbers are fixed and all
// consecutive numbers are
// distinct
using System;

class GFG
{

// Returns the total ways to
// form arrays such that every
// consecutive element is
// different and each element
// except the first and last
// can take values from 1 to M
static int totalWays(int N,
                     int M, int X)
{

    // define the dp[][] array
    int [,]dp = new int[N + 1, 2];

    // if the first element is 1
    if (X == 1)
    {

        // there is only one
        // way to place a 1
        // at the first index
        dp[0, 0] = 1;
    }
    else
    {
        // the value at first index
        // needs to be 1, thus there
        // is no way to place a
        // non-one integer
        dp[0, 1] = 0;
    }

    // if the first element was 1
    // then at index 1, only non
    // one integer can be placed
    // thus there are M - 1 ways
    // to place a non one integer
    // at index 2 and 0 ways to
    // place a 1 at the 2nd index
    if (X == 1)
    {
        dp[1, 0] = 0;
        dp[1, 1] = M - 1;
    }

    // Else there is one way to
    // place a one at index 2
    // and if a non one needs to
    // be placed here, there are
    // (M - 2) options, i.e neither
    // the element at this index
    // should be 1, neither should
    // it be equal to the previous
    // element
    else
    {
        dp[1, 0] = 1;
        dp[1, 1] = (M - 2);
    }

    // Build the dp array
    // in bottom up manner
    for (int i = 2; i < N; i++)
    {

        // f(i, one) = f(i - 1,
        // non-one)
        dp[i, 0] = dp[i - 1, 1];

        // f(i, non-one) =
        // f(i - 1, one) * (M - 1) +
        // f(i - 1, non-one) * (M - 2)
        dp[i, 1] = dp[i - 1, 0] * (M - 1) +
                dp[i - 1, 1] * (M - 2);
    }

    // last element needs to be
    // one, so return dp[n - 1][0]
    return dp[N - 1, 0];
}

// Driver Code
public static void Main ()
{
    int N = 4, M = 3, X = 2;
    Console.WriteLine(totalWays(N, M, X));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// PHP program to count the
// number of ways to form
// arrays of N numbers such
// that the first and last
// numbers are fixed and all
// consecutive numbers are distinct

// Returns the total ways to
// form arrays such that every
// consecutive element is
// different and each element
// except the first and last
// can take values from 1 to M
function totalWays($N, $M, $X)
{

    // define the dp[][] array
    $dp = array(array());

    // if the first element is 1
    if ($X == 1)
    {

        // there is only one
        // way to place a 1
        // at the first index
        $dp[0][0] = 1;
    }
    else
    {

        // the value at first
        // index needs to be 1,
        // thus there is no way
        // to place a non-one integer
        $dp[0][1] = 0;
    }

    // if the first element was
    // 1 then at index 1, only
    // non one integer can be
    // placed thus there are M - 1
    // ways to place a non one
    // integer at index 2 and 0 ways
    // to place a 1 at the 2nd index
    if ($X == 1)
    {
        $dp[1][0] = 0;
        $dp[1][1] = $M - 1;
    }

    // Else there is one way
    // to place a one at index
    // 2 and if a non one needs
    // to be placed here, there
    // are (M - 2) options, i.e
    // neither the element at
    // this index should be 1,
    // neither should it be equal
    // to the previous element
    else
    {
        $dp[1][0] = 1;
        $dp[1][1] = ($M - 2);
    }

    // Build the dp array
    // in bottom up manner
    for ($i = 2; $i <$N; $i++)
    {

        // f(i, one) =
        // f(i - 1, non-one)
        $dp[$i][0] = $dp[$i - 1][1];
```

```
        // f(i, non-one) =
        // f(i - 1, one) * (M - 1) +
        // f(i - 1, non-one) * (M - 2)
        $dp[$i][1] = $dp[$i - 1][0] *
                           ($M - 1) +
                     $dp[$i - 1][1] *
                           ($M - 2);
    }

    // last element needs to
    // be one, so return dp[n - 1][0]
    return $dp[$N - 1][0];
}

// Driver Code
$N = 4; $M = 3; $X = 2;
echo totalWays($N, $M, $X);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
3
```

Time Complexity: O(N), where N is the size of the array

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-of-ways-to-form-an-array-with-distinct-adjacent-elements/

Chapter 95

Number of ways to get even sum by choosing three numbers from 1 to N

Number of ways to get even sum by choosing three numbers from 1 to N - GeeksforGeeks

Given an integer N, find the number of ways we can choose 3 numbers from {1, 2, 3 …, N} such that their sum is even.
Examples:

```
Input :  N = 3
Output : 1
Explanation: Select 1, 2 and 3

Input :  N = 4
Output :  2
Either select (1, 2, 3) or (1, 3, 4)
```

Chapter 96

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

To get sum even there can be only 2 cases:

1. Take 2 odd numbers and 1 even.
2. Take all even numbers.

```
If n is even,
  Count of odd numbers = n/2 and even = n/2.
Else
  Count odd numbers = n/2 +1 and evne = n/2.
```

Case 1 – No. of ways will be : ${}^{odd}C_2$ * even.
Case 2 – No. of ways will be : ${}^{even}C_3$.

So, total ways will be Case_1_result + Case_2_result.

C++

```
 // C++ program for above implementation
#include <bits/stdc++.h>
#define MOD 1000000007
using namespace std;

// Function to count number of ways
int countWays(int N)
{
    long long int count, odd = N / 2, even;
    if (N & 1)
        odd = N / 2 + 1;

    even = N / 2;

    // Case 1: 2 odds and 1 even
    count = (((odd * (odd - 1)) / 2) * even) % MOD;

    // Case 2: 3 evens
    count = (count + ((even * (even - 1) *
                       (even - 2)) / 6)) % MOD;

    return count;
}

// Driver code
int main()
{
    int n = 10;
    cout << countWays(n) << endl;
    return 0;
}
```

Java

```
 // java program for above implementation
import java.io.*;

class GFG {

    static long MOD = 1000000007;

    // Function to count number of ways
    static long countWays(int N)
    {
        long count, odd = N / 2, even;

        if ((N & 1) > 0)
            odd = N / 2 + 1;

        even = N / 2;

        // Case 1: 2 odds and 1 even
        count = (((odd * (odd - 1)) / 2)
                          * even) % MOD;

        // Case 2: 3 evens
        count = (count + ((even * (even
                - 1) * (even - 2)) / 6))
                                 % MOD;

        return (long)count;
    }

    // Driver code
    static public void main (String[] args)
    {
        int n = 10;

        System.out.println(countWays(n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python3 code for above implementation

MOD = 1000000007

# Function to count number of ways
def countWays( N ):
    odd = N / 2
    if N & 1:
        odd = N / 2 + 1
    even = N / 2

    # Case 1: 2 odds and 1 even
    count = (((odd * (odd - 1)) / 2) * even) % MOD

    # Case 2: 3 evens
    count = (count + ((even * (even - 1) *
            (even - 2)) / 6)) % MOD
    return count

# Driver code
n = 10
print(int(countWays(n)))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
 // C# program for above implementation
```

```
using System;

public class GFG {

    static long MOD = 1000000007;

    // Function to count number of ways
    static long countWays(int N)
    {
        long count, odd = N / 2, even;

        if ((N & 1) > 0)
            odd = N / 2 + 1;

        even = N / 2;

        // Case 1: 2 odds and 1 even
        count = (((odd * (odd - 1)) / 2)
                            * even) % MOD;

        // Case 2: 3 evens
        count = (count + ((even * (even
                  - 1) * (even - 2)) / 6))
                                    % MOD;

        return (long)count;
    }

    // Driver code
    static public void Main ()
    {
        int n = 10;

        Console.WriteLine(countWays(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for
// above implementation

$MOD = 1000000007;

// Function to count
// number of ways
function countWays($N)
{
    global $MOD;

    $count;
    $odd =$N / 2;
    $even;
    if ($N & 1)
        $odd = $N / 2 + 1;

    $even = $N / 2;

    // Case 1: 2 odds
    // and 1 even
    $count = ((($odd * ($odd - 1)) / 2) *
                            $even) % $MOD;

    // Case 2: 3 evens
    $count = ($count + (($even * ($even - 1) *
                        ($even - 2)) / 6)) % $MOD;

    return $count;
}

    // Driver Code
    $n = 10;
    echo countWays($n);

// This code is contributed by anuj_67.
?>
```

Output:

```
60
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-ways-get-even-sum-choosing-three-numbers-1-n/

Chapter 97

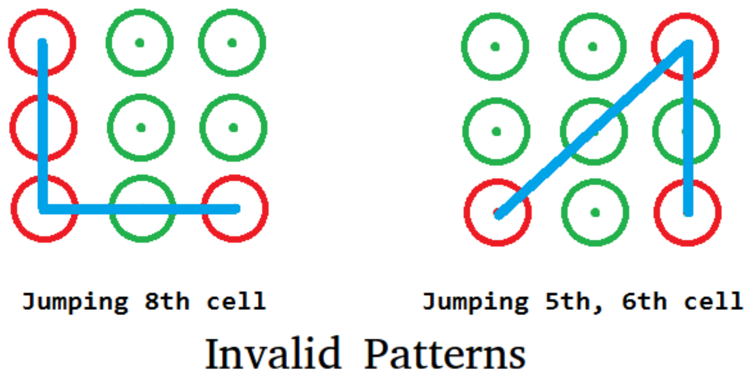
Number of ways to make mobile lock pattern

Number of ways to make mobile lock pattern - GeeksforGeeks

A mobile pattern is a grid of 3X3 cell, where drawing a specific pattern (connecting specific sequence of cells in order) will unlock the mobile. In this problem, the task is to calculate number of ways of making the lock pattern with number of connections in given range. In general terms, we are given a range as min and max, we need to tell how many patterns can be made which use at least min connection cell and at most max connection cell.

```
Input  : min = 5, max = 7
Output : 106080
Below are some example patterns
```

A simple solution is to do simple DFS by going through various connections from all starting points. We can optimize by seeing the symmetry between patterns, we can treat cell 1, 3, 7 and 9 as same. Similarly, we can treat 2, 4, 6 and 8 as same. Answer returned by a member of same group can be multiplied by 4 to get result by all members.
Next thing to note is that below patterns are not valid, because jumping some cell is not allowed as in below diagram cell 8 and cell 5, 6 are jumped.



To take care of such invalid moves, a 2D jump array is taken in below code which stores possible jump cells in jump array. When we call recursively we impose an extra condition that if we are moving from one cell to another which involves some jumping cell, then this cell should be visited already to ignore invalid moves.
In below code we have made recursive calls from 1, 2 and 5 only because 1 will cover 3, 5 and 7 and 2 will cover 4, 6 and 8 due to symmetry.
See below code for better understanding :

```
 //  C/C++ program to find number of ways to lock the mobile
// pattern
#include <bits/stdc++.h>
using namespace std;
#define DOTS 10

//  method to find total pattern starting from current cell
int totalPatternFromCur(bool visit[DOTS], int jump[DOTS][DOTS],
                                          int cur, int toTouch)
{
    if (toTouch <= 0)
    {
        //  if last cell then return 1 way
        return (toTouch == 0)? 1 : 0;
    }

    int ways = 0;

    //  make this cell visited before going to next call
    visit[cur] = true;

    for (int i=1; i<DOTS; i++)
    {
       /*  if this cell is not visit AND
           either i and cur are adjacent (then jump[i][cur] = 0)
           or between cell must be visit already (
           then visit[jump[i][cur]] = 1)   */
        if (!visit[i] && (!jump[i][cur] || visit[jump[i][cur]]))
          ways += totalPatternFromCur(visit, jump, i, toTouch - 1);
    }

    //  make this cell not visited after returning from call
    visit[cur] = false;

    return ways;
}

//  method returns number of pattern with minimum m connection
// and maximum n connection
int waysOfConnection(int m, int n)
{
    int jump[DOTS][DOTS] = {0};

    //  2 lies between 1 and 3
    jump[1][3] = jump[3][1] = 2;

    //  8 lies between 7 and 9
    jump[7][9] = jump[9][7] = 8;

    //  4 lies between 1 and 7
    jump[1][7] = jump[7][1] = 4;

    //  6 lies between 3 and 9
    jump[3][9] = jump[9][3] = 6;

    //  5 lies between 1, 9  2, 8  3, 7 and 4, 6
    jump[1][9] = jump[9][1] = jump[2][8] = jump[8][2]
     = jump[3][7] = jump[7][3] = jump[4][6] = jump[6][4] = 5;

    bool visit[DOTS] = {0};
    int ways = 0;
    for (int i = m; i <= n; i++)
    {
        //  1, 3, 7, 9 are symmetric so multiplying by 4
        ways += 4 * totalPatternFromCur(visit, jump, 1, i - 1);

        //  2, 4, 6, 8 are symmetric so multiplying by 4
        ways += 4 * totalPatternFromCur(visit, jump, 2, i - 1);

        ways += totalPatternFromCur(visit, jump, 5, i - 1);
    }

    return ways;
}
```

```
//  Driver code to test above method
int main()
{
    int minConnect = 4;
    int maxConnect = 6;

    cout << waysOfConnection(minConnect, maxConnect);

    return 0;
}
```

Output:

```
34792
```

Source

https://www.geeksforgeeks.org/number-of-ways-to-make-mobile-lock-pattern/

Chapter 98

Number of ways to merge two arrays such retaining order

Number of ways to merge two arrays such retaining order - GeeksforGeeks

Given two array of size **n** and m. The task is to find the number of ways we can merge the given arrays into one array such that order of elements of each array doesn't change.

Examples:

```
Input : n = 2, m = 2
Output : 6
Let first array of size n = 2 be [1, 2] and second array of size m = 2 be [3, 4].
So, possible merged array of n + m elements can be:
[1, 2, 3, 4]
[1, 3, 2, 4]
[3, 4, 1, 2]
[3, 1, 4, 2]
[1, 3, 4, 2]
[3, 1, 2, 4]

Input : n = 4, m = 6
Output : 210
```

The idea is to use the concept of combinatorics. Suppose we have two array A{a1, a2, …., am} and B{b1, b2, …., bn} having m and n elements respectively and now we have to merge them without loosing their order.
After merging we know that the total number of element will be (m + n) element after merging. So, now we just need the ways to choose m places out of (m + n) where you will place element of array A in its actual order, which is ${}^{m + n}C_n$.
After placing m element of array A, n spaces will be left, which can be filled by the n elements of B array in its actual order.

So, total number of ways to merge two array such that their order in merged array is same is $^{\mathbf{m+n}}\mathbf{C_n}$

Below is the implementation of this appraoch:

C++

```
 // CPP Program to find number of ways
// to merge two array such that their
// order in merged array is same
#include <bits/stdc++.h>
using namespace std;

// function to find the binomial coefficient
int binomialCoeff(int n, int k)
{
    int C[k + 1];
    memset(C, 0, sizeof(C));

    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++) {

        // Compute next row of pascal triangle
        // using the previous row
        for (int j = min(i, k); j > 0; j--)
            C[j] = C[j] + C[j - 1];
    }
    return C[k];
}

// function to find number of ways
// to merge two array such that their
// order in merged array is same
int numOfWays(int n, int m)
{
    return binomialCoeff(m + n, m);
}

// Driven Program
int main()
{
    int n = 2, m = 2;
    cout << numOfWays(n, m) << endl;
    return 0;
}
```

Java

```
 // Java Program to find number of ways
// to merge two array such that their
// order in merged array is same

import java.io.*;

class GFG {

    // function to find the binomial
    // coefficient
    static int binomialCoeff(int n, int k)
    {
        int C[] = new int[k + 1];
        // memset(C, 0, sizeof(C));

        C[0] = 1; // nC0 is 1

        for (int i = 1; i <= n; i++) {

            // Compute next row of pascal
            // triangle using the previous
            // row
            for (int j = Math.min(i, k);
                                 j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }

        return C[k];
    }

    // function to find number of ways
    // to merge two array such that their
    // order in merged array is same
    static int numOfWays(int n, int m)
    {
        return binomialCoeff(m + n, m);
    }

    // Driven Program
    public static void main (String[] args)
    {
        int n = 2, m = 2;
        System.out.println(numOfWays(n, m));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# Program to find number of ways
// to merge two array such that their
// order in merged array is same

using System;

class GFG {

    // function to find the binomial
    // coefficient
    static int binomialCoeff(int n, int k)
    {
        int []C = new int[k + 1];
        // memset(C, 0, sizeof(C));

        C[0] = 1; // nC0 is 1

        for (int i = 1; i <= n; i++) {

            // Compute next row of pascal
            // triangle using the previous
            // row
            for (int j = Math.Min(i, k);
                            j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }

        return C[k];
    }

    // function to find number of ways
    // to merge two array such that their
    // order in merged array is same
    static int numOfWays(int n, int m)
    {
        return binomialCoeff(m + n, m);
    }

    // Driven Program
    public static void Main ()
    {
        int n = 2, m = 2;
        Console.WriteLine(numOfWays(n, m));
    }
}
```

```
// This code is contributed by anuj_67.
```

Output:

```
6
```

We can solve above problem in linear time using linear time implementation of binomial coefficient.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-ways-merge-two-arrays-retaining-order/

Chapter 99

Number of ways to reach Nth floor by taking at-most K leaps

Number of ways to reach Nth floor by taking at-most K leaps - GeeksforGeeks

Given N number of stairs. Also given the number of steps that one can cover at most in one leap (K). The task is to find the number of possible ways one (**only consider combinations**) can climb to the top of the building in K leaps or less from the ground floor.

Examples:

> **Input:** N = 5, K = 3
> **Output:** 5
> To reach stair no-5 we can choose following combination of leaps:
> 1 1 1 1 1
> 1 1 1 2
> 1 2 2
> 1 1 3
> 2 3
> Therefore the answer is 5.
>
> **Input:** N = 29, K = 5
> **Output:** 603

Let *combo[i]* be the number of ways to reach the i-th floor. Hence the number of ways to reach combo[i] from combo[j] by taking a leap of i-j will be ***combo[i] += combo[j].*** So iterate for all possible leaps, and for each possible leaps keep adding the possible combinations to the combo array. The final answer will be stored in combo[N].

Below is the implementation of the above approach.

C++

```
 // C++ program to reach N-th stair
// by taking a maximum of K leap
#include <bits/stdc++.h>

using namespace std;

int solve(int N, int K)
{

    // elements of combo[] stores the no of
    // possible ways to reach it by all
    // combinations of k leaps or less

    int combo[N + 1] = { 0 };

    // assuming leap 0 exist and assigning
    // its value to 1 for calculation
    combo[0] = 1;

    // loop to iterate over all
    // possible leaps upto k;
    for (int i = 1; i <= K; i++) {

        // in this loop we count all possible
        // leaps to reach the jth stair with
        // the help of ith leap or less
        for (int j = 0; j <= N; j++) {

            // if the leap is not more than the i-j
            if (j >= i) {

                // calculate the value and
                // store in combo[j]
                // to reuse it for next leap
                // calculation for the jth stair
                combo[j] += combo[j - i];
            }
        }
    }

    // returns the no of possible number
    // of leaps to reach the top of
    // building of n stairs
    return combo[N];
}

// Driver Code
int main()
```

```
{
    // N i the no of total stairs
    // K is the value of the greatest leap
    int N = 29;
    int K = 5;

    cout << solve(N, K);

    solve(N, K);
    return 0;
}
```

Java

```
 // Java program to reach N-th
// stair by taking a maximum
// of K leap
class GFG
{
static int solve(int N, int K)
{

    // elements of combo[] stores
    // the no. of possible ways
    // to reach it by all combinations
    // of k leaps or less
    int[] combo;
    combo = new int[50];

    // assuming leap 0 exist
    // and assigning its value
    // to 1 for calculation
    combo[0] = 1;

    // loop to iterate over all
    // possible leaps upto k;
    for (int i = 1; i <= K; i++)
    {

        // in this loop we count all
        // possible leaps to reach
        // the jth stair with the
        // help of ith leap or less
        for (int j = 0; j <= N; j++)
        {

            // if the leap is not
            // more than the i-j
            if (j >= i)
            {

                // calculate the value and
                // store in combo[j] to
                // reuse it for next leap
                // calculation for the
                // jth stair
                combo[j] += combo[j - i];
            }
        }
    }

    // returns the no of possible
    // number of leaps to reach
    // the top of building of
    // n stairs
    return combo[N];
}

// Driver Code
public static void main(String args[])
{
    // N i the no of total stairs
    // K is the value of the
    // greatest leap
    int N = 29;
    int K = 5;

    System.out.println(solve(N, K));

    solve(N, K);
}
}

// This code is contributed
// by ankita_saini
```

C#

```
 // C# program to reach N-th
// stair by taking a maximum
// of K leap
using System;

class GFG
{
static int solve(int N, int K)
```

```
{

    // elements of combo[] stores
    // the no. of possible ways
    // to reach it by all combinations
    // of k leaps or less
    int[] combo;
    combo = new int[50];

    // assuming leap 0 exist
    // and assigning its value
    // to 1 for calculation
    combo[0] = 1;

    // loop to iterate over all
    // possible leaps upto k;
    for (int i = 1; i <= K; i++)
    {

        // in this loop we count all
        // possible leaps to reach
        // the jth stair with the
        // help of ith leap or less
        for (int j = 0; j <= N; j++)
        {

            // if the leap is not
            // more than the i-j
            if (j >= i)
            {

                // calculate the value and
                // store in combo[j] to
                // reuse it for next leap
                // calculation for the
                // jth stair
                combo[j] += combo[j - i];
            }
        }
    }

    // returns the no of possible
    // number of leaps to reach
    // the top of building of
    // n stairs
    return combo[N];
}
```

```
// Driver Code
public static void Main()
{
    // N i the no of total stairs
    // K is the value of the
    // greatest leap
    int N = 29;
    int K = 5;

    Console.WriteLine(solve(N, K));
    solve(N, K);
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
 <?php
error_reporting(0);
// PHP program to reach N-th
// stair by taking a maximum
// of K leap
function solve($N, $K)
{

    // elements of combo[] stores
    // the no of possible ways to
    // reach it by all combinations
    // of k leaps or less
    $combo[$N + 1] = array();

    // assuming leap 0 exist and
    // assigning its value to 1
    // for calculation
    $combo[0] = 1;

    // loop to iterate over all
    // possible leaps upto k;
    for ($i = 1; $i <= $K; $i++)
    {

        // in this loop we count
        // all possible leaps to
        // reach the jth stair with
        // the help of ith leap or less
        for ($j = 0; $j <= $N; $j++)
        {
            // if the leap is not
            // more than the i-j
            if ($j >= $i)
            {

                // calculate the value and
                // store in combo[j]
                // to reuse it for next leap
                // calculation for the jth stair
                $combo[$j] += $combo[$j - $i];
            }
        }
    }

    // returns the no of possible
    // number of leaps to reach
    // the top of building of n stairs
    return $combo[$N];
}

// Driver Code

// N i the no of total stairs
// K is the value of the greatest leap
$N = 29;
$K = 5;

echo solve($N, $K);

solve($N, $K);

// This code is contributed
// by Akanksha Rai(Abby_akku)
?>
```

Output:

```
603
```

Time Complexity: O(N*K)
Auxiliary Space: O(N)

Improved By : ankita_saini, Abby_akku

Source

https://www.geeksforgeeks.org/number-of-ways-to-reach-nth-floor-by-taking-at-most-k-leaps/

Chapter 100

Palindromic Selfie Numbers

Palindromic Selfie Numbers - GeeksforGeeks

Given a number x, find it's palindromic selfie number according to selfie multiplicative rule. If such a number doesn't exist, then print "No such number exists".

A Palindromic selfie number satisfies the selfie multiplicative rule such that there exists another number y with *x * reverse_digits_of(x) = y * reverse_digits_of(y)*, with the condition that the number y is obtained by some ordering of the digits in x, i.e x and y should have same digits with different order.

Examples :

```
Input : 1224
Output : 2142
Explanation :
Because, 1224 X 4221 = 2142 X 2412
And all digits of 2142 are formed by a different
permutation of the digits in 1224
(Note: The valid output is either be 2142 or 2412)

Input : 13452
Output : 14532
Explanation :
Because, 13452 X 25431 = 14532 X 23541
And all digits of 14532 are formed by a different
permutation of the digits in 13452

Input : 12345
Output : No such number exists
Explanation :
Because, with no combination of digits 1, 2, 3, 4, 5
could we get a number that satisfies
12345 X 54321 = number X reverse_of_its_digits
```

Approach :

- The idea is to break down the number and obtain all permutations of the digits in the number.
- Then, the number and its palindrome are removed from the set of permutations obtained, which will form the LHS of our equality.
- To check for RHS, we now iterate over all other permutations equating

  ```
  LHS = current_number X palindrome(current_number)
  ```

- As soon as we get a match, we exit the loop with an affirmative message, else print "No such number available".

Below is Java implementation of above approach :

```
 // Java program to find palindromic selfie numbers
import java.util.*;

public class palindrome_selfie {
    // To store all permuations of digits in the number
    Set<Integer> all_permutes = new HashSet<Integer>();

    int number; // input number

    public palindrome_selfie(int num)
    {
        number = num;
    }

    // Function to reverse the digits of a number
    public int palindrome(int num)
    {
        int reversednum = 0;
        int d;
        while (num > 0) {
            d = num % 10; // Extract last digit

            // Append it at the beg
            reversednum = reversednum * 10 + d;
            num = num / 10;  // Reduce number until 0
        }

        return reversednum;
    }

    // Function to check palindromic selfie
```

```
public void palin_selfie()
{
    // Length of the number required for
    // calculating all permuatations of the digits
    int l = String.valueOf(number).length() - 1;

    this.permute(number, 0, l); // Calculate all permuations

    /* Remove the number and its palindrome from
       the obtained set as this is the LHS of
       multiplicative equality */
    all_permutes.remove(palindrome(number));
    all_permutes.remove(number);

    boolean flag = false; // Denotes the status result

    // Iterate over all other numbers
    Iterator it = all_permutes.iterator();
    while (it.hasNext()) {
        int number2 = (int)it.next();

        // Check for equality x*palin(x) = y*palin(y)
        if (number * palindrome(number) ==
                     number2 * palindrome(number2)) {
            System.out.println("Palindrome multiplicative" +
                                "selfie of "+ number + " is  : "
                                 + number2);

            flag = true; // Answer found
            break;
        }
    }

    // If no such number found
    if (flag == false) {
        System.out.println("Given number has no palindrome selfie.");
    }
}

// Function to get all possible possible permuations
// of the digits in num
public void permute(int num, int l, int r)
{
    // Adds the new permuation obatined in the set
    if (l == r)
        all_permutes.add(num);

    else {
            for (int i = l; i <= r; i++) {

                // Swap digits to get a different ordering
                num = swap(num, l, i);

                // Recurse to next pair of digits
                permute(num, l + 1, r);
                num = swap(num, l, i); // Swap back
            }
        }
    }

    // Function that swaps the digits i and j in the num
    public int swap(int num, int i, int j)
    {
        char temp;

        // Convert int to char array
        char[] charArray = String.valueOf(num).toCharArray();

        // Swap the ith and jth character
        temp = charArray[i];
        charArray[i] = charArray[j];
        charArray[j] = temp;

        // Convert back to int and return
        return Integer.valueOf(String.valueOf(charArray));
    }

    // Driver Function
    public static void main(String args[])
    {
        // First example, input = 145572
        palindrome_selfie example1 = new palindrome_selfie(145572);
        example1.palin_selfie();

        // Second example, input = 19362
        palindrome_selfie example2 = new palindrome_selfie(19362);
        example2.palin_selfie();

        // Third example, input = 4669
        palindrome_selfie example3 = new palindrome_selfie(4669);
        example3.palin_selfie();
    }
}
```

Output :

```
Palindrome multiplicative selfie of 145572 is  : 157452
Given number has no palindrome selfie.
Palindrome multiplicative selfie of 4669 is  : 6496
```

Source

https://www.geeksforgeeks.org/palindromic-selfie-numbers/

Chapter 101

Paper Cut into Minimum Number of Squares

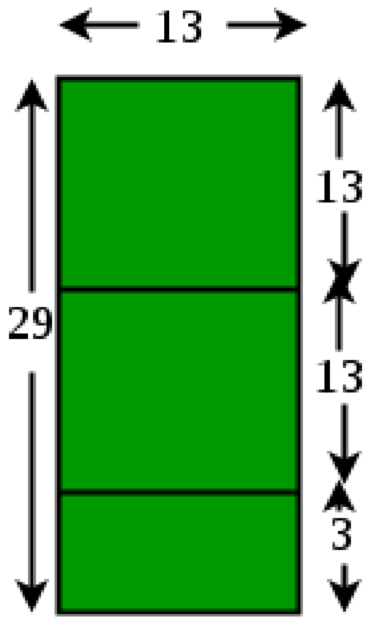
Paper Cut into Minimum Number of Squares - GeeksforGeeks

Given a paper of size A x B. Task is to cut the paper into squares of any size. Find the minimum number of squares that can be cut from the paper.

Examples:

```
Input  : 13 x 29
Output : 9
Explanation :
2 (squares of size 13x13) +
4 (squares of size 3x3) +
3 (squares of size 1x1)=9

Input  : 4 x 5
Output : 5
Explanation :
1 (squares of size 4x4) +
4 (squares of size 1x1)
```

We know that if we want to cut minimum number of squares from the paper then we would have to cut largest square possible from the paper first and largest square will have same side as smaller side of the paper. For example if paper have the size 13 x 29, then maximum square will be of side 13. so we can cut 2 square of size 13 x 13 (29/13 = 2). Now remaining paper will have size 3 x 13. Similarly we can cut remaining paper by using 4 squares of size 3 x 3 and 3 squares of 1 x 1. So minimum 9 squares can be cut from the Paper of size 13 x 29.



Below is the implementation of above approach.

C++

```
 // C++ program to find minimum number of squares
// to cut a paper.
#include<bits/stdc++.h>
using namespace std;

// Returns min number of squares needed
int minimumSquare(int a, int b)
{
    long long result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater then 0
    while (b > 0)
    {
        // Update result
        result += a/b;

        long long rem = a % b;
        a = b;
        b = rem;
    }

    return result;
}

// Driver code
int main()
{
    int n = 13, m = 29;
    cout << minimumSquare(n, m);
    return 0;
}
```

Java

```
 // Java program to find minimum
// number of squares to cut a paper.
class GFG{

// To swap two numbers
static void swap(int a,int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Returns min number of squares needed
static int minimumSquare(int a, int b)
{
    int result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater then 0
    while (b > 0)
    {
        // Update result
        result += a/b;
        rem = a % b;
        a = b;
        b = rem;
    }
    return result;
}

// Driver code
public static void main(String[] args)
{
    int n = 13, m = 29;
    System.out.println(minimumSquare(n, m));
}
}

//This code is contributed by Smitha Dinesh Semwal.
```

Python3

```
 # Python 3 program to find minimum
# number of squares to cut a paper.

# Returns min number of squares needed
def minimumSquare(a, b):

    result = 0
    rem = 0

    # swap if a is small size side .
    if (a < b):
        a, b = b, a

    # Iterate until small size side is
    # greater then 0
    while (b > 0):

        # Update result
        result += int(a / b)

        rem = int(a % b)
        a = b
        b = rem

    return result

# Driver code
n = 13
m = 29

print(minimumSquare(n, m))

# This code is contributed by
# Smitha Dinesh Semwal
```

Output:

```
9
```

Note that the above Greedy solution doesn't always produce optimal result. For example if input is 36 x 30, the above algorithm would produce output 6, but we can cut the paper in 5 squares
1) Three squares of size 12 x 12
2) Two squares of size 18 x 18.

Thanks to Sergey V. Pereslavtsev for pointing out the above case.

Source

https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/

Chapter 102

Permutation and Combination in Python

Permutation and Combination in Python - GeeksforGeeks

Python provide direct methods to find permutations and combinations of a sequence. These methods are present in itertools package.

Permutation

First import itertools package to implement permutations method in python. This method takes a list as an input and return an object list of tuples that contain all permutation in a list form.

```
 # A Python program to print all
# permutations using library function
from itertools import permutations

# Get all permutations of [1, 2, 3]
perm = permutations([1, 2, 3])

# Print the obtained permutations
for i in list(perm):
    print i
```

Output

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

It generates n! permutations if length of input sequence is n.

If want want to get permutations of length L then implement it in this way.

```
 # A Python program to print all
# permutations of given length
from itertools import permutations

# Get all permutations of length 2
# and length 2
perm = permutations([1, 2, 3], 2)

# Print the obtained permutations
for i in list(perm):
    print i
```

Output

```
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

It generate nCr * r! permutations if length of input sequence is n and input parameter is r.

Combination

This method takes a list and a input r as a input and return a object list of tuples which contain all possible combination of length r in a list form.

```
 # A Python program to print all
# combinations of given length
from itertools import combinations

# Get all combinations of [1, 2, 3]
# and length 2
comb = combinations([1, 2, 3], 2)

# Print the obtained combinations
for i in list(comb):
    print i
```

Output

```
(1, 2)
(1, 3)
(2, 3)
```

1. Combinations are emitted in lexicographic sort order of input. So, if the input list is sorted, the combination tuples will be produced in sorted order.

   ```
    # A Python program to print all combinations
   # of given length with unsorted input.
   from itertools import combinations

   # Get all combinations of [2, 1, 3]
   # and length 2
   comb = combinations([2, 1, 3], 2)

   # Print the obtained combinations
   for i in list(comb):
       print i
   ```

   Output

   ```
   (2, 1)
   (2, 3)
   (1, 3)
   ```

2. Elements are treated as unique based on their position, not on their value. So if the input elements are unique, there will be no repeat values in each combination.

   ```
    # A Python program to print all combinations
   # of given length with duplicates in input
   from itertools import combinations

   # Get all combinations of [1, 1, 3]
   # and length 2
   comb = combinations([1, 1, 3], 2)

   # Print the obtained combinations
   for i in list(comb):
       print i
   ```

   Output

   ```
   (1, 1)
   (1, 3)
   (1, 3)
   ```

3. If we want to make combination of same element to same element then we use combinations_with_replacement.

```
 # A Python program to print all combinations
# with an element-to-itself combination is
# also included
from itertools import combinations_with_replacement

# Get all combinations of [1, 2, 3] and length 2
comb = combinations_with_replacement([1, 2, 3], 2)

# Print the obtained combinations
for i in list(comb):
    print i
```

Output

```
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)
```

Source

https://www.geeksforgeeks.org/permutation-and-combination-in-python/

Chapter 103

Permutations of a given string using STL

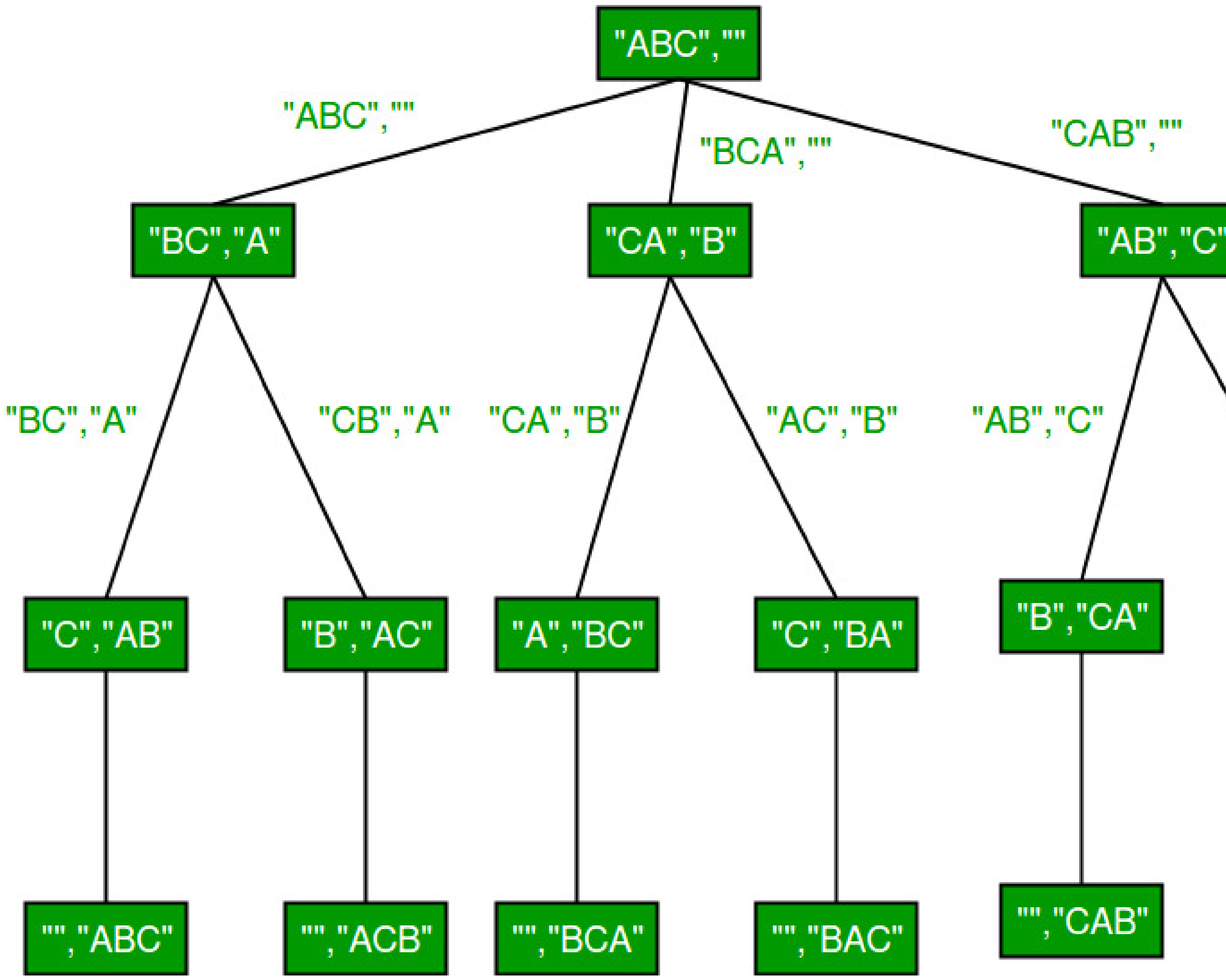
Permutations of a given string using STL - GeeksforGeeks

A permutation, also called an "arrangement number" or "order", is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! permutation.
Source: Mathword

Below are the permutations of string ABC.
ABC ACB BAC BCA CBA CAB
We have discussed C implementation to print all permutations of a given string using backtracking here. In this post, C++ implementation using STL is discussed.

Method 1 (Using rotate())

std::rotate function rotates elements of a vector/string such that the passed middle element becomes first. For example, if we call rotate for "ABCD" with middle as second element, the string becomes "BCDA" and if we again call rotate with middle as second element, the string becomes "CDAB". Refer this for a sample program.



Below is C++ implementation.

```
 // C++ program to print all permutations with
// duplicates allowed using rotate() in STL
#include <bits/stdc++.h>
using namespace std;

// Function to print permutations of string str,
// out is used to store permutations one by one
void permute(string str, string out)
{
    // When size of str becomes 0, out has a
    // permutation (length of out is n)
    if (str.size() == 0)
    {
        cout << out << endl;
        return;
    }

    // One be one move all characters at
    // the beginning of out (or result)
    for (int i = 0; i < str.size(); i++)
    {
        // Remove first character from str and
        // add it to out
        permute(str.substr(1), out + str[0]);

        // Rotate string in a way second character
        // moves to the beginning.
        rotate(str.begin(), str.begin() + 1, str.end());
    }
}

// Driver code
int main()
{
    string str = "ABC";
    permute(str, "");
    return 0;
}
```

Output :

```
ABC
ACB
BCA
BAC
CAB
CBA
```

Method 2 (using next_permute())

We can use next_permute() that modifies a string so that it stores lexicographically next permutation. If current string is lexicographically largest, i.e., "CBA", then next_permute() returns false.

We first sort the string, so that it is converted to lexicographically smallest permutation. Then we one by one call next_permutation until it returns false.

```
 // C++ program to print all permutations with
```

```
// duplicates allowed using next_permute()
#include <bits/stdc++.h>
using namespace std;

// Function to print permutations of string str
// using next_permute()
void permute(string str)
{
    // Sort the string in lexicographically
    // ascennding order
    sort(str.begin(), str.end());

    // Keep printing next permutation while there
    // is next permutation
    do {
       cout << str << endl;
    } while (next_permutation(str.begin(), str.end()));
}

// Driver code
int main()
{
    string str = "CBA";
    permute(str);
    return 0;
}
```

Output :

```
ABC
ACB
BCA
BAC
CAB
CBA
```

Note that the second method always prints permutations in lexicographically sorted order irrespective of input string.

Source

https://www.geeksforgeeks.org/permutations-of-a-given-string-using-stl/

Chapter 104

Print all distinct permutations of a given string with duplicates

Print all distinct permutations of a given string with duplicates - GeeksforGeeks

Given a string that may contain duplicates, write a function to print all permutations of given string such that no permutation is repeated in output.

Examples:

```
Input:  str[] = "AB"
Output: AB BA

Input:  str[] = "AA"
Output: AA

Input:  str[] = "ABC"
Output: ABC ACB BAC BCA CBA CAB

Input:  str[] = "ABA"
Output: ABA AAB BAA

Input:  str[] = "ABCA"
Output: AABC AACB ABAC ABCA ACBA ACAB BAAC BACA
        BCAA CABA CAAB CBAA
```

We have discussed an algorithm to print all permutations in below post. It is strongly recommended to refer below post as a prerequisite of this post.

Write a C program to print all permutations of a given string

The algorithm discussed on above link doesn't handle duplicates.

```
 // Program to print all permutations of a string in sorted order.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Following function is needed for library function qsort(). */
int compare(const void *a, const void * b)
{
    return ( *(char *)a - *(char *)b );
}

// A utility function two swap two characters a and b
void swap(char* a, char* b)
{
    char t = *a;
    *a = *b;
    *b = t;
}

// This function finds the index of the smallest character
// which is greater than 'first' and is present in str[l..h]
int findCeil(char str[], char first, int l, int h)
{
    // initialize index of ceiling element
    int ceilIndex = l;

    // Now iterate through rest of the elements and find
    // the smallest character greater than 'first'
    for (int i = l+1; i <= h; i++)
        if (str[i] > first && str[i] < str[ceilIndex])
            ceilIndex = i;

    return ceilIndex;
}

// Print all permutations of str in sorted order
void sortedPermutations(char str[])
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort(str, size, sizeof( str[0] ), compare);

    // Print permutations one by one
    bool isFinished = false;
    while (!isFinished)
    {
        // print this permutation
        static int x = 1;
        printf("%d  %s \n", x++, str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for (i = size - 2; i >= 0; --i)
            if (str[i] < str[i+1])
                break;

        // If there is no such chracter, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if (i == -1)
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
            int ceilIndex = findCeil(str, str[i], i + 1, size - 1);

            // Swap first and second characters
            swap(&str[i], &str[ceilIndex]);

            // Sort the string on right of 'first char'
            qsort(str + i + 1, size - i - 1, sizeof(str[0]), compare);
        }
    }
}

// Driver program to test above function
int main()
{
    char str[] = "ACBC";
    sortedPermutations( str );
    return 0;
}
```

Output:

```
1  ABCC
2  ACBC
3  ACCB
4  BACC
5  BCAC
6  BCCA
7  CABC
8  CACB
```

```
9  CBAC
10  CBCA
11  CCAB
12  CCBA
```

The above code is taken from a comment below by Mr. Lazy.

Time Complexity: O(n^2 * n!)
Auxiliary Space: O(1)

Source

https://www.geeksforgeeks.org/print-all-permutations-of-a-string-with-duplicates-allowed-in-input-string/

Chapter 105

Print all permutations in sorted (lexicographic) order

Print all permutations in sorted (lexicographic) order - GeeksforGeeks

Given a string, print all permutations of it in sorted order. For example, if the input string is "ABC", then output should be "ABC, ACB, BAC, BCA, CAB, CBA".

We have discussed a program to print all permutations in this post, but here we must print the permutations in increasing order.

Following are the steps to print the permutations lexicographic-ally

1. Sort the given string in non-decreasing order and print it. The first permutation is always the string sorted in non-decreasing order.

2. Start generating next higher permutation. Do it until next higher permutation is not possible. If we reach a permutation where all characters are sorted in non-increasing order, then that permutation is the last permutation.

Steps to generate the next higher permutation:
1. Take the previously printed permutation and find the rightmost character in it, which is smaller than its next character. Let us call this character as 'first character'.

2. Now find the ceiling of the 'first character'. Ceiling is the smallest character on right of 'first character', which is greater than 'first character'. Let us call the ceil character as 'second character'.

3. Swap the two characters found in above 2 steps.

4. Sort the substring (in non-decreasing order) after the original index of 'first character'.

Let us consider the string "ABCDEF". Let previously printed permutation be "DCFEBA". The next permutation in sorted order should be "DEABCF". Let us understand above steps to find next permutation. The 'first character' will be 'C'. The 'second character' will be 'E'. After swapping these two, we get "DEFCBA". The final step is to sort the substring after the character original index of 'first character'. Finally, we get "DEABCF".

Following is C++ implementation of the algorithm.

```
 // Program to print all permutations of a string in sorted order.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Following function is needed for library function qsort(). Refer
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare (const void *a, const void * b)
{  return ( *(char *)a - *(char *)b ); }

// A utility function two swap two characters a and b
void swap (char* a, char* b)
{
    char t = *a;
    *a = *b;
    *b = t;
}

// This function finds the index of the smallest character
// which is greater than 'first' and is present in str[l..h]
int findCeil (char str[], char first, int l, int h)
{
    // initialize index of ceiling element
    int ceilIndex = l;

    // Now iterate through rest of the elements and find
    // the smallest character greater than 'first'
    for (int i = l+1; i <= h; i++)
      if (str[i] > first && str[i] < str[ceilIndex])
            ceilIndex = i;

    return ceilIndex;
}

// Print all permutations of str in sorted order
void sortedPermutations ( char str[] )
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort( str, size, sizeof( str[0] ), compare );

    // Print permutations one by one
    bool isFinished = false;
    while ( ! isFinished )
    {
        // print this permutation
        printf ("%s \n", str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for ( i = size - 2; i >= 0; --i )
           if (str[i] < str[i+1])
              break;

        // If there is no such chracter, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if ( i == -1 )
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
            int ceilIndex = findCeil( str, str[i], i + 1, size - 1 );

            // Swap first and second characters
            swap( &str[i], &str[ceilIndex] );

            // Sort the string on right of 'first char'
            qsort( str + i + 1, size - i - 1, sizeof(str[0]), compare );
        }
    }
}

// Driver program to test above function
int main()
{
    char str[] = "ABCD";
    sortedPermutations( str );
    return 0;
}
```

Output:

```
ABCD
ABDC
....
....
DCAB
DCBA
```

The upper bound on time complexity of the above program is O(n^2 x n!). We can optimize step 4 of the above algorithm for finding next permutation. Instead of sorting the subarray after the 'first character', we can reverse the subarray, because the subarray we get after swapping is always sorted in non-increasing order. This optimization makes the time complexity as O(n x n!). See following optimized code.

```
 // An optimized version that uses reverse instead of sort for
// finding the next permutation

// A utility function to reverse a string str[l..h]
void reverse(char str[], int l, int h)
{
   while (l < h)
   {
       swap(&str[l], &str[h]);
       l++;
       h--;
   }
}

// Print all permutations of str in sorted order
void sortedPermutations ( char str[] )
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort( str, size, sizeof( str[0] ), compare );

    // Print permutations one by one
    bool isFinished = false;
    while ( ! isFinished )
    {
        // print this permutation
        printf ("%s \n", str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for ( i = size - 2; i >= 0; --i )
           if (str[i] < str[i+1])
              break;

        // If there is no such chracter, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if ( i == -1 )
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
            int ceilIndex = findCeil( str, str[i], i + 1, size - 1 );

            // Swap first and second characters
            swap( &str[i], &str[ceilIndex] );

            // reverse the string on right of 'first char'
            reverse( str, i + 1, size - 1 );
        }
    }
}
```

The above programs print duplicate permutation when characters are repeated. We can avoid it by keeping track of the previous permutation. While printing, if the current permutation is same as previous permutation, we won't print it.

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/lexicographic-permutations-of-string/

Chapter 106

Print all permutations with repetition of characters

Print all permutations with repetition of characters - GeeksforGeeks

Given a string of length n, print all permutation of the given string. Repetition of characters is allowed. Print these permutations in lexicographically sorted order
Examples:

```
Input: AB
Ouput: All permutations of AB with repetition are:
      AA
      AB
      BA
      BB

Input: ABC
Output: All permutations of ABC with repetition are:
       AAA
       AAB
       AAC
       ABA
       ...
       ...
       CCB
       CCC
```

For an input string of size n, there will be n^n permutations with repetition allowed. The idea is to fix the first character at first index and recursively call for other subsequent indexes. Once all permutations starting with the first character are printed, fix the second character at first index. Continue these steps till last character. Thanks to PsychoCoder for providing following C implementation.

C

```
 # C program to print all permutations with repetition
# of characters
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

/* Following function is used by the library qsort() function
  to sort an array of chars */
int compare (const void * a, const void * b);

/* The main function that recursively prints all repeated
   permutations of  the given string. It uses data[] to store all
   permutations one by one */
void allLexicographicRecur (char *str, char* data, int last, int index)
{
    int i, len = strlen(str);

    // One by one fix all characters at the given index and recur for
    // the/ subsequent indexes
    for ( i=0; i<len; i++ )
    {
        // Fix the ith character at index and if this is not the last
        // index then recursively call for higher indexes
        data[index] = str[i] ;

        // If this is the last index then print the string stored in
        // data[]
        if (index == last)
            printf("%s\n", data);
        else // Recur for higher indexes
            allLexicographicRecur (str, data, last, index+1);
    }
}

/* This function sorts input string, allocate memory for data (needed
   for allLexicographicRecur()) and calls allLexicographicRecur() for
   printing all  permutations */
void allLexicographic(char *str)
{
    int len = strlen (str) ;

    // Create a temp array that will be used by allLexicographicRecur()
    char *data = (char *) malloc (sizeof(char) * (len + 1)) ;
    data[len] = '\0';

    // Sort the input string so that we get all output strings in
    // lexicographically sorted order
    qsort(str, len, sizeof(char), compare);

    // Now print all permutaions
    allLexicographicRecur (str, data, len-1, 0);

    // Free data to avoid memory leak
    free(data);
}

// Needed for library function qsort()
int compare (const void * a, const void * b)
{
    return ( *(char *)a - *(char *)b );
}

// Driver program to test above functions
int main()
{
    char str[] = "ABC";
    printf("All permutations with repetition of %s are: \n",
            str);
    allLexicographic(str);
    return 0;
}
```

Python

```
 # Python program to print all permutations with repetition
# of characters

def toString(List):
    return ''.join(List)

# The main function that recursively prints all repeated
# permutations of the given string. It uses data[] to store
# all permutations one by one
def allLexicographicRecur (string, data, last, index):
    length = len(string)

    # One by one fix all characters at the given index and
    # recur for the subsequent indexes
    for i in xrange(length):

        # Fix the ith character at index and if this is not
        # the last index then recursively call for higher
        # indexes
        data[index] = string[i]
        # If this is the last index then print the string
        # stored in data[]
        if index==last:
            print toString(data)
        else:
            allLexicographicRecur(string, data, last, index+1)

# This function sorts input string, allocate memory for data
# (needed for allLexicographicRecur()) and calls
# allLexicographicRecur() for printing all permutations
def allLexicographic(string):
    length = len(string)

    # Create a temp array that will be used by
    # allLexicographicRecur()
    data = [""] * (length+1)

    # Sort the input string so that we get all output strings in
    # lexicographically sorted order
    string = sorted(string)

    # Now print all permutaions
    allLexicographicRecur(string, data, length-1, 0)

# Driver program to test the above functions
string = "ABC"
print "All permutations with repetition of " + string + " are:"
allLexicographic(string)

# This code is contributed to Bhavya Jain
```

Following is recursion tree for input string “AB”. The purpose of recursion tree is to help in understanding the above implementation as it shows values of different variables.

```
                  data=""
              /         \
             /           \
       index=0           index=0
        i=0               i=1
      data="A"           data="B"
       /   \              /    \
     /      \            /      \
  index=1  index=1    index=1    index=1
   i=0      i=1        i=0        i=1
data="AA"  data="AB"  data="BA"  data="BB"
```

In the above implementation, it is assumed that all characters of the input string are different. The implementation can be easily modified to handle the repeated characters. We have to add a preprocessing step to find unique characters (before calling allLexicographicRecur()).

Source

https://www.geeksforgeeks.org/print-all-permutations-with-repetition-of-characters/

Chapter 107

Print all possible strings of length k that can be formed from a set of n characters

Print all possible strings of length k that can be formed from a set of n characters - GeeksforGeeks

Given a set of characters and a positive integer k, print all possible strings of length k that can be formed from the given set.

Examples:

```
Input:
set[] = {'a', 'b'}, k = 3

Output:
aaa
aab
aba
abb
baa
bab
bba
bbb


Input:
set[] = {'a', 'b', 'c', 'd'}, k = 1
Output:
a
b
```

```
c
d
```

For a given set of size n, there will be n^k possible strings of length k. The idea is to start from an empty output string (we call it *prefix* in following code). One by one add all characters to *prefix*. For every character added, print all possible strings with current prefix by recursively calling for k equals to k-1.
Below is the implementation of above idea :

Java

```
 // Java program to print all
// possible strings of length k

class GFG {

// The method that prints all
// possible strings of length k.
// It is mainly a wrapper over
// recursive function printAllKLengthRec()
static void printAllKLength(char[] set, int k)
{
    int n = set.length;
    printAllKLengthRec(set, "", n, k);
}

// The main recursive method
// to print all possible
// strings of length k
static void printAllKLengthRec(char[] set,
                               String prefix,
                               int n, int k)
{

    // Base case: k is 0,
    // print prefix
    if (k == 0)
    {
        System.out.println(prefix);
        return;
    }

    // One by one add all characters
    // from set and recursively
    // call for k equals to k-1
    for (int i = 0; i < n; ++i)
    {
        // Next character of input added
        String newPrefix = prefix + set[i];

        // k is decreased, because
        // we have added a new character
        printAllKLengthRec(set, newPrefix,
                                n, k - 1);
    }
}

// Driver Code
public static void main(String[] args)
{
    System.out.println("First Test");
    char[] set1 = {'a', 'b'};
    int k = 3;
    printAllKLength(set1, k);

    System.out.println("\nSecond Test");
    char[] set2 = {'a', 'b', 'c', 'd'};
    k = 1;
    printAllKLength(set2, k);
}
}
```

C#

```
 // C# program to print all
// possible strings of length k
using System;

class GFG {

// The method that prints all
// possible strings of length k.
// It is mainly a wrapper over
// recursive function printAllKLengthRec()
static void printAllKLength(char[] set, int k)
{
    int n = set.Length;
    printAllKLengthRec(set, "", n, k);
}

// The main recursive method
// to print all possible
// strings of length k
static void printAllKLengthRec(char[] set,
                               String prefix,
                              int n, int k)
{

    // Base case: k is 0,
    // print prefix
    if (k == 0)
    {
        Console.WriteLine(prefix);
        return;
    }

    // One by one add all characters
    // from set and recursively
    // call for k equals to k-1
    for (int i = 0; i < n; ++i)
    {

        // Next character of input added
        String newPrefix = prefix + set[i];

        // k is decreased, because
        // we have added a new character
        printAllKLengthRec(set, newPrefix,
                                n, k - 1);
    }
}

// Driver Code
static public void Main ()
{
    Console.WriteLine("First Test");
    char[] set1 = {'a', 'b'};
    int k = 3;
    printAllKLength(set1, k);

    Console.WriteLine("\nSecond Test");
    char[] set2 = {'a', 'b', 'c', 'd'};
    k = 1;
    printAllKLength(set2, k);
}
}

// This code is contributed by Ajit.
```

Output:

```
First Test
aaa
```

```
aab
aba
abb
baa
bab
bba
bbb

Second Test
a
b
c
d
```

The above solution is mainly generalization of this post.

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

Improved By : jit_t

Source

https://www.geeksforgeeks.org/print-all-combinations-of-given-length/

Chapter 108

Print all possible strings that can be made by placing spaces

Print all possible strings that can be made by placing spaces - GeeksforGeeks

Given a string you need to print all possible strings that can be made by placing spaces (zero or one) in between them
Examples :

```
Input :  str[] = "ABC"
Output : ABC
         AB C
         A BC
         A B C

Input : str[] = "ABCD"
Output : ABCD
         A BCD
         AB CD
         A B CD
         ABC D
         A BC D
         AB C D
         A B C D
```

If we take a closer look, we can notice that this problem boils down to Power Set problem. We basically need to generate all subsets where every element is a different space.

C++

```
 // C++ program to print all strings that can be
```

```
// made by placing spaces
#include <bits/stdc++.h>
using namespace std;

void printSubsequences(string str)
{
    int n = str.length();
    unsigned int opsize = pow(2, n - 1);

    for (int counter = 0; counter < opsize; counter++) {
        for (int j = 0; j < n; j++) {

            cout << str[j];
            if (counter & (1 << j))
                cout << " ";
        }
        cout << endl;
    }
}

// Driver code
int main()
{
    string str = "ABC";
    printSubsequences(str);
    return 0;
}
```

Java

```
 // Java program to print all strings that can be
// made by placing spaces
import java.util.*;
class GFG
{
static void printSubsequences(String s)
{
    char[] str= s.toCharArray();
    int n = str.length;
    int opsize = (int)(Math.pow(2, n - 1));

    for (int counter = 0; counter < opsize; counter++) {
        for (int j = 0; j < n; j++) {

            System.out.print(str[j]);
            if ((counter & (1 << j)) > 0)
                System.out.print(" ");
        }
        System.out.println();
    }
}

// Driver code
public static void main(String[] args)
{
    String str = "AB";
    printSubsequences(str);
}
}

/* This code is contributed by Mr. Somesh Awasthi */
```

C#

```
 // C# program to print all strings
// that can be made by placing spaces
using System;

class GFG {

    // Function to print all subsequences
    static void printSubsequences(String s)
    {
        char[] str= s.ToCharArray();
        int n = str.Length;
        int opsize = (int)(Math.Pow(2, n - 1));

        for (int counter = 0; counter < opsize;
                                     counter++)
        {
            for (int j = 0; j < n; j++)
            {
                Console.Write(str[j]);

                if ((counter & (1 << j)) > 0)
                    Console.Write(" ");
            }
            Console.WriteLine();
        }
    }

    // Driver code
    public static void Main()
    {
        String str = "ABC";
        printSubsequences(str);
    }
}

// This code is contributed by shiv_bhakt.
```

PHP

```
 <?php
// PHP program to print
// all strings that can be
// made by placing spaces

function printSubsequences($str)
{
    $n = strlen($str);
    $opsize = pow(2, $n - 1);

    for ($counter = 0;
         $counter < $opsize;
         $counter++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            echo $str[$j];
            if ($counter & (1 << $j))
                echo " ";
        }
        echo "\n";
    }
}

// Driver code
$str = "ABC";
printSubsequences($str);

// This code is contributed by ajit
?>
```

Output :

```
ABC
A BC
AB C
A B C
```

Asked in: Amazon

Improved By : shiv_bhakt, jit_t

Source

https://www.geeksforgeeks.org/print-possible-strings-can-made-placing-spaces-2/

Chapter 109

Print all subsets of given size of a set

Print all subsets of given size of a set - GeeksforGeeks

Generate all possible subset of size **r** of given array with distinct elements.

Examples:

```
Input  : arr[] = {1, 2, 3, 4}
            r = 2
Output :  1 2
          1 3
          1 4
          2 3
          2 4
          3 4

Input  : arr[] = {10, 20, 30, 40, 50}
            r = 3
Output : 10 20 30
         10 20 40
         10 20 50
         10 30 40
         10 30 50
         10 40 50
         20 30 40
         20 30 50
         20 40 50
         30 40 50
```

This problem is same as Print all possible combinations of r elements in a given array of size n.

The idea here is similar to Subset Sum Problem. We one by one consider every element of input array, and recur for two cases:

1) The element is included in current combination (We put the element in data[] and increment next available index in data[])
2) The element is excluded in current combination (We do not put the element and do not change index)

When number of elements in data[] become equal to r (size of a combination), we print it.

This method is mainly based on Pascal's Identity, i.e. $\mathbf{n_{c_r} = n\text{-}1_{c_r} + n\text{-}1_{c_{r-1}}}$

C++

```
 // Program to print all combination of size r in
// an array of size n
#include <stdio.h>
void combinationUtil(int arr[], int n, int r,
                     int index, int data[], int i);

// The main function that prints all combinations of
// size r in arr[] of size n. This function mainly
// uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination
    // one by one
    int data[r];

    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, n, r, 0, data, 0);
}

/* arr[]  ---> Input Array
   n      ---> Size of input array
   r      ---> Size of a combination to be printed
   index  ---> Current index in data[]
   data[] ---> Temporary array to store current combination
   i      ---> index of current element in arr[]     */
void combinationUtil(int arr[], int n, int r, int index,
                     int data[], int i)
{
    // Current cobination is ready, print it
    if (index == r) {
        for (int j = 0; j < r; j++)
            printf("%d ", data[j]);
        printf("\n");
        return;
    }

    // When no more elements are there to put in data[]
    if (i >= n)
        return;

    // current is included, put next at next location
    data[index] = arr[i];
    combinationUtil(arr, n, r, index + 1, data, i + 1);

    // current is excluded, replace it with next
    // (Note that i+1 is passed, but index is not
    // changed)
    combinationUtil(arr, n, r, index, data, i + 1);
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 20, 30, 40, 50 };
    int r = 3;
    int n = sizeof(arr) / sizeof(arr[0]);
    printCombination(arr, n, r);
    return 0;
}
```

Java

```
 // Java program to print all combination of size
// r in an array of size n
import java.io.*;

class Permutation {

    /* arr[]  ---> Input Array
    data[] ---> Temporary array to store current combination
    start & end ---> Staring and Ending indexes in arr[]
    index  ---> Current index in data[]
    r ---> Size of a combination to be printed */
    static void combinationUtil(int arr[], int n, int r,
                          int index, int data[], int i)
    {
        // Current combination is ready to be printed,
        // print it
        if (index == r) {
            for (int j = 0; j < r; j++)
                System.out.print(data[j] + " ");
            System.out.println("");
            return;
        }

        // When no more elements are there to put in data[]
        if (i >= n)
            return;

        // current is included, put next at next
        // location
        data[index] = arr[i];
        combinationUtil(arr, n, r, index + 1,
                               data, i + 1);

        // current is excluded, replace it with
        // next (Note that i+1 is passed, but
        // index is not changed)
        combinationUtil(arr, n, r, index, data, i + 1);
    }

    // The main function that prints all combinations
    // of size r in arr[] of size n. This function
    // mainly uses combinationUtil()
    static void printCombination(int arr[], int n, int r)
    {
        // A temporary array to store all combination
        // one by one
        int data[] = new int[r];

        // Print all combination using temprary
        // array 'data[]'
        combinationUtil(arr, n, r, 0, data, 0);
    }

    /* Driver function to check for above function */
    public static void main(String[] args)
    {
        int arr[] = { 10, 20, 30, 40, 50 };
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}
/* This code is contributed by Devesh Agrawal */
```

Python3

```
 # Python program to print all
# subset combination of n
# element in given set of r element .

# arr[] ---> Input Array
# data[] ---> Temporary array to
#             store current combination
# start & end ---> Staring and Ending
#                  indexes in arr[]
# index ---> Current index in data[]
# r ---> Size of a combination
#       to be printed
def combinationUtil(arr, n, r,
                    index, data, i):
    # Current combination is
    # ready to be printed,
    # print it
    if(index == r):
        for j in range(r):
            print(data[j], end = " ")
        print(" ")
        return

    # When no more elements
    # are there to put in data[]
    if(i >= n):
        return

    # current is included,
    # put next at next
    # location
    data[index] = arr[i]
    combinationUtil(arr, n, r,
                    index + 1, data, i + 1)

    # current is excluded,
    # replace it with
    # next (Note that i+1
    # is passed, but index
    # is not changed)
    combinationUtil(arr, n, r, index,
                    data, i + 1)


# The main function that
# prints all combinations
# of size r in arr[] of
# size n. This function
# mainly uses combinationUtil()
def printcombination(arr, n, r):
    # A temporary array to
    # store all combination
    # one by one
    data = list(range(r))

    # Print all combination
    # using temporary
    # array 'data[]'
    combinationUtil(arr, n, r,
                    0, data, 0)


# Driver Code
arr = [10, 20, 30, 40, 50]

r = 3
n = len(arr)
printcombination(arr, n, r)

# This code is contributed
# by Ambuj sahu
```

C#

```
 // C# program to print all combination
// of size r in an array of size n
using System;

class GFG {

    /* arr[] ---> Input Array
    data[] ---> Temporary array to store
    current combination start & end --->
    Staring and Ending indexes in arr[]
    index ---> Current index in data[]
    r ---> Size of a combination to be
    printed */
    static void combinationUtil(int []arr,
                  int n, int r, int index,
                          int []data, int i)
    {

        // Current combination is ready to
        // be printed, print it
        if (index == r)
        {
            for (int j = 0; j < r; j++)
```

```
                Console.Write(data[j] + " ");

            Console.WriteLine("");

            return;
        }

        // When no more elements are there
        // to put in data[]
        if (i >= n)
            return;

        // current is included, put next
        // at next location
        data[index] = arr[i];
        combinationUtil(arr, n, r, index + 1,
                                data, i + 1);

        // current is excluded, replace
        // it with next (Note that i+1
        // is passed, but index is not
        // changed)
        combinationUtil(arr, n, r, index,
                                data, i + 1);
    }

    // The main function that prints all
    // combinations of size r in arr[] of
    // size n. This function mainly uses
    // combinationUtil()
    static void printCombination(int []arr,
                                int n, int r)
    {

        // A temporary array to store all
        // combination one by one
        int []data = new int[r];

        // Print all combination using
        // temprary array 'data[]'
        combinationUtil(arr, n, r, 0, data, 0);
    }

    /* Driver function to check for
    above function */
    public static void Main()
    {
        int []arr = { 10, 20, 30, 40, 50 };
        int r = 3;
        int n = arr.Length;

        printCombination(arr, n, r);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Program to print all combination of
// size r in an array of size n

// The main function that prints all
// combinations of size r in arr[] of
// size n. This function mainly uses
// combinationUtil()
function printCombination( $arr, $n, $r)
{
    // A temporary array to store all
    // combination one by one
    $data = array();

    // Print all combination using
    // temprary array 'data[]'
    combinationUtil($arr, $n, $r, 0, $data, 0);
}

/* arr[] ---> Input Array
n ---> Size of input array
r ---> Size of a combination to be printed
index ---> Current index in data[]
data[] ---> Temporary array to store
current combination
i ---> index of current element in arr[] */
function combinationUtil( $arr, $n, $r, $index,
                    $data, $i)
{
    // Current cobination is ready, print it
    if ($index == $r) {
        for ( $j = 0; $j < $r; $j++)
            echo $data[$j]," ";
        echo "\n";
        return;
    }
    // When no more elements are there to
    // put in data[]
    if ($i >= $n)
        return;

    // current is included, put next at
    // next location
    $data[$index] = $arr[$i];
    combinationUtil($arr, $n, $r, $index + 1,
                              $data, $i + 1);

    // current is excluded, replace it with
    // next (Note that i+1 is passed, but
    // index is not changed)
    combinationUtil($arr, $n, $r, $index,
                            $data, $i + 1);
}

// Driver program to test above functions
    $arr = array( 10, 20, 30, 40, 50 );
    $r = 3;
    $n = count($arr);
    printCombination($arr, $n, $r);

// This code is contributed by anuj_67.
?>
```

Output:

```
10 20 30
10 20 40
10 20 50
10 30 40
10 30 50
10 40 50
20 30 40
20 30 50
20 40 50
30 40 50
```

Refer below post for more solutions and ideas to handle duplicates in input array.
Print all possible combinations of r elements in a given array of size n.

Improved By : vt_m, AmbujSahu

Source

https://www.geeksforgeeks.org/print-subsets-given-size-set/

Chapter 110

Print all the combinations of N elements by changing sign such that their sum is divisible by M

Print all the combinations of N elements by changing sign such that their sum is divisible by M - GeeksforGeeks

Given an array of N integers and an integer M. You can change the sign(positive or negative) of any element in the array. The task is to print all possible combinations of the array elements that can be obtained by changing the sign of the elements such that their sum is divisible by M.

Note: You have to take all of the array elements in each combination and in the same order as the elements present in the array. However, you can change the sign of elements.

Examples:

> **Input:** a[] = {5, 6, 7}, M = 3
> **Output:**
> -5-6-7
> +5-6+7
> -5+6-7
> +5+6+7
>
> **Input:** a[] = {3, 5, 6, 8}, M = 5
> **Output:**
> -3-5+6-8
> -3+5+6-8
> +3-5-6+8
> +3+5-6+8

Approach: The concept of power set is used here to solve this problem. Using power-set generate all possible combinations of signs that can be applied to the array of elements. If the sum obtained is divisible by M, then print the combination. Below are the steps:

- Iterate for all possible combinations of '+' and '-' using power set.
- Iterate on the array elements and if the j-th bit from left is set, then assume the array element to be positive and if the bit is not set, then assume the array element to be negative. Refer here to check if bit any index is set or not.
- If the sum is divisible by M, then the again traverse the array elements and print them along with sign('+' or '-').

Below is the implementation of the above approach:

C++

```
 #include <bits/stdc++.h>
using namespace std;

// Function to print all the combinations
void printCombinations(int a[], int n, int m)
{

    // Iterate for all combinations
    for (int i = 0; i < (1 << n); i++) {
        int sum = 0;

        // Initially 100 in binary if n is 3
        // as 1<<(3-1) = 100 in binary
        int num = 1 << (n - 1);

        // Iterate in the array and assign signs
        // to the array elements
        for (int j = 0; j < n; j++) {

            // If the j-th bit from left is set
            // take '+' sign
            if (i & num)
                sum += a[j];
            else
                sum += (-1 * a[j]);

            // Right shift to check if
            // jth bit is set or not
            num = num >> 1;
        }

        if (sum % m == 0) {

            // re-initialize
            num = 1 << (n - 1);
            // Iterate in the array elements
            for (int j = 0; j < n; j++) {

                // If the jth from left is set
                if ((i & num))
                    cout << "+" << a[j] << " ";
                else
                    cout << "-" << a[j] << " ";

                // right shift
                num = num >> 1;
            }
            cout << endl;
        }
    }
}

// Driver Code
int main()
{
    int a[] = { 3, 5, 6, 8 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = 5;

    printCombinations(a, n, m);
    return 0;
}
```

Java

```
 import java.io.*;

class GFG
{

// Function to print
// all the combinations
static void printCombinations(int a[],
                              int n, int m)
{

    // Iterate for all
    // combinations
    for (int i = 0;
             i < (1 << n); i++)
    {
        int sum = 0;
        // Initially 100 in binary
        // if n is 3 as
        // 1<<(3-1) = 100 in binary
        int num = 1 << (n - 1);

        // Iterate in the array
        // and assign signs to
        // the array elements
        for (int j = 0; j < n; j++)
        {

            // If the j-th bit
            // from left is set
            // take '+' sign
            if ((i & num) > 0)
                sum += a[j];
            else
                sum += (-1 * a[j]);

            // Right shift to check if
            // jth bit is set or not
            num = num >> 1;
        }

        if (sum % m == 0)
        {

            // re-initialize
            num = 1 << (n - 1);

            // Iterate in the
            // array elements
            for (int j = 0; j < n; j++)
            {

                // If the jth from
                // left is set
                if ((i & num) > 0)
                    System.out.print("+" +
                                     a[j] + " ");
                else
                    System.out.print("-" +
                                     a[j] + " ");

                // right shift
                num = num >> 1;
            }
        System.out.println();
        }
    }
}

// Driver code
public static void main(String args[])
{
    int a[] = { 3, 5, 6, 8 };
    int n = a.length;
    int m = 5;

    printCombinations(a, n, m);
}
}

// This code is contributed
// by inder_verma.
```

Python3

```
 # Function to print
# all the combinations
def printCombinations(a, n, m):

    # Iterate for all
    # combinations
    for i in range(0, (1 << n)):

        sum = 0

        # Initially 100 in binary
        # if n is 3 as
        # 1<<(3-1) = 100 in binary
        num = 1 << (n - 1)

        # Iterate in the array
        # and assign signs to
        # the array elements
        for j in range(0, n):

            # If the j-th bit
            # from left is set
            # take '+' sign
            if ((i & num) > 0):
                sum += a[j]
            else:
                sum += (-1 * a[j])
            # Right shift to check if
            # jth bit is set or not
            num = num >> 1

        if (sum % m == 0):

            # re-initialize
            num = 1 << (n - 1)

            # Iterate in the
            # array elements
            for j in range(0, n):

                # If the jth from
                # left is set
                if ((i & num) > 0):
                    print("+", a[j], end = " ",
                                     sep = "")
                else:
                    print("-", a[j], end = " ",
                                     sep = "")

                # right shift
                num = num >> 1
            print("")

# Driver code
a = [ 3, 5, 6, 8 ]
n = len(a)
m = 5
printCombinations(a, n, m)

# This code is contributed
# by smita.
```

C#

```
 // Print all the combinations
// of N elements by changing
// sign such that their sum
// is divisible by M
using System;

class GFG
{

// Function to print
// all the combinations
static void printCombinations(int []a,
                              int n, int m)
{

    // Iterate for all
    // combinations
    for (int i = 0;
             i < (1 << n); i++)
    {
        int sum = 0;

        // Initially 100 in binary
        // if n is 3 as
        // 1<<(3-1) = 100 in binary
        int num = 1 << (n - 1);

        // Iterate in the array
        // and assign signs to
        // the array elements
        for (int j = 0; j < n; j++)
        {

            // If the j-th bit
            // from left is set
            // take '+' sign
            if ((i & num) > 0)
                sum += a[j];
            else
                sum += (-1 * a[j]);

            // Right shift to check if
            // jth bit is set or not
            num = num >> 1;
        }

        if (sum % m == 0)
        {

            // re-initialize
            num = 1 << (n - 1);

            // Iterate in the
            // array elements
            for (int j = 0; j < n; j++)
            {

                // If the jth from
                // left is set
                if ((i & num) > 0)
                    Console.Write("+" +
                              a[j] + " ");
                else
                    Console.Write("-" +
                              a[j] + " ");

                // right shift
                num = num >> 1;
            }

        Console.Write("\n");
        }
    }
}

// Driver code
public static void Main()
{
    int []a = { 3, 5, 6, 8 };
    int n = a.Length;
    int m = 5;

    printCombinations(a, n, m);
}
}

// This code is contributed
// by Smitha.
```

Output:

```
-3 -5 +6 -8
-3 +5 +6 -8
+3 -5 -6 +8
+3 +5 -6 +8
```

Time Complexity: O(2^N * N), where N is the number of elements.

Improved By : inderDuMCA, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-all-the-combinations-of-n-elements-by-changing-sign-such-that-their-sum-is-d

Chapter 111

Print all the palindromic permutations of given string in alphabetic order

Print all the palindromic permutations of given string in alphabetic order - GeeksforGeeks

Given a string **str** of size **n**. The problem is to print all the palindromic permutations of **str** in alphabetic order if possible else print "-1".

Examples :

```
Input : str = "aabb"
Output :
abba
baab

Input : malayalam
Output :
aalmymlaa
aamlylmaa
alamymala
almayamla
amalylama
amlayalma
laamymaal
lamayamal
lmaayaaml
maalylaam
malayalam
mlaayaalm
```

Prerequisites: Lexicographically first palindromic string and Next higher palindromic number using the same set of digits.

Approach: Following are the steps:

1. If possible, find the Lexicographically first palindromic string using the characters of **str** else print "-1" and return.
2. If lexicographically first palindromic string is obtained then print it.
3. Now, find the next higher palindromic string using the same set of characters of **str**. Refer this post.
   The only difference between the two posts is that in one digits are being arranged and in the other lowercase characters are being arranged.
4. If next higher palindromic string is obtained then print it and goto step 3 else return.

Note that the string **str** is always manipulated so as to get the palindromic strings using the steps mentioned above.

C++

```
 // C++ implementation to print all the palindromic
// permutations alphabetically
#include <bits/stdc++.h>
using namespace std;

const char MAX_CHAR = 26;

// Function to count frequency of each char in the
// string. freq[0] for 'a', ...., freq[25] for 'z'
void countFreq(char str[], int freq[], int n)
{
    for (int i = 0; i < n; i++)
        freq[str[i] - 'a']++;
}

// Cases to check whether a palindromic
// string can be formed or not
bool canMakePalindrome(int freq[], int n)
{
    // count_odd to count no of
    // chars with odd frequency
    int count_odd = 0;
    for (int i = 0; i < MAX_CHAR; i++)
        if (freq[i] % 2 != 0)
            count_odd++;

    // For even length string
    // no odd freq character
    if (n % 2 == 0) {
        if (count_odd > 0)
            return false;
        else
            return true;
    }

    // For odd length string
    // one odd freq character
    if (count_odd != 1)
        return false;

    return true;
}

// Function to find odd freq char and reducing its
// freq by 1, returns garbage value if odd freq
// char is not present
char findOddAndRemoveItsFreq(int freq[])
{
    char odd_char;

    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] % 2 != 0) {
            freq[i]--;
            odd_char = (char)(i + 'a');
            break;
        }
    }

    return odd_char;
}

// To find lexicographically first palindromic
// string.
bool findPalindromicString(char str[], int n)
{
    int freq[MAX_CHAR] = { 0 };
    countFreq(str, freq, n);

    // check whether a palindromic string
    // can be formed or not with the
    // characters of 'str'
    if (!canMakePalindrome(freq, n))
        // cannot be formed
        return false;

    // Assigning odd freq character if present
    // else some garbage value
    char odd_char = findOddAndRemoveItsFreq(freq);

    // indexes to store characters from the front
    // and end
    int front_index = 0, rear_index = n - 1;

    // Traverse characters in alphabetical order
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] != 0) {
            char ch = (char)(i + 'a');

            // store 'ch' to half its frequency times
            // from the front and rear end. Note that
            // odd character is removed by
            // findOddAndRemoveItsFreq()
            for (int j = 1; j <= freq[i] / 2; j++) {
                str[front_index++] = ch;
                str[rear_index--] = ch;
            }
        }
    }

    // if true then odd frequency char exists
    // store it to its corresponding index
    if (front_index == rear_index)
        str[front_index] = odd_char;

    // palindromic string can be formed
    return true;
}

// function to reverse the characters in the
// range i to j in 'str'
void reverse(char str[], int i, int j)
{
    while (i < j) {
        swap(str[i], str[j]);
        i++;
        j--;
    }
}

// function to find next higher palindromic
// string using the same set of characters
bool nextPalin(char str[], int n)
{
    // if length of 'str' is less than '3'
    // then no higher palindromic string
    // can be formed
    if (n <= 3)
        return false;

    // find the index of last character
    // in the 1st half of 'str'
    int mid = n / 2 - 1;
    int i, j;

    // Start from the (mid-1)th character and
    // find the the first character that is
    // alphabetically smaller than the
    // character next to it.
    for (i = mid - 1; i >= 0; i--)
        if (str[i] < str[i + 1])
            break;

    // If no such character is found, then all characters
    // are in descending order (alphabetcally) which
    // means there cannot be a higher palindromic string
    // with same set of characters
    if (i < 0)
        return false;

    // Find the alphabetically smallest character on
    // right side of ith character which is greater
    // than str[i] up to index 'mid'
    int smallest = i + 1;
    for (j = i + 2; j <= mid; j++)
        if (str[j] > str[i] && str[j] < str[smallest])
            smallest = j;

    // swap str[i] with str[smallest]
    swap(str[i], str[smallest]);

    // as the string is a palindrome, the same
    // swap of characters should be performed
    // in the 2nd half of 'str'
    swap(str[n - i - 1], str[n - smallest - 1]);

    // reverse characters in the range (i+1) to mid
    reverse(str, i + 1, mid);

    // if n is even, then reverse characters in the
    // range mid+1 to n-i-2
    if (n % 2 == 0)
        reverse(str, mid + 1, n - i - 2);
    // else if n is odd, then reverse characters
    // in the range mid+2 to n-i-2
    else
        reverse(str, mid + 2, n - i - 2);

    // next alphabetically higher palindromic
    // string can be formed
    return true;
}

// function to print all the palindromic permutations
// alphabetically
void printAllPalinPermutations(char str[], int n)
{
    // check if lexicographically first palindromic string
    // can be formed or not using the characters of 'str'
    if (!(findPalindromicString(str, n))) {
        // cannot be formed
        cout << "-1";
        return;
    }

    // print all palindromic permutations
    do {
        cout << str << endl;
    } while (nextPalin(str, n));
}

// Driver program to test above
int main()
{
    char str[] = "malayalam";
    int n = strlen(str);
    printAllPalinPermutations(str, n);
    return 0;
}
```

Java

```
 // Java code to print all the
// palindromic permutations alphabetically
import java.util.*;

class GFG {

static int MAX_CHAR = 26;
```

```
// Function to count frequency
// of each char in the string.
// freq[0] for 'a', ...., freq[25] for 'z'
static void countFreq(char str[],
                    int freq[], int n){

    for (int i = 0; i < n; i++)
        freq[str[i] - 'a']++;
}

// Cases to check whether a palindromic
// string can be formed or not
static Boolean canMakePalindrome
            (int freq[], int n){

    // count_odd to count no of
    // chars with odd frequency
    int count_odd = 0;

    for (int i = 0; i < MAX_CHAR; i++)
        if (freq[i] % 2 != 0)
            count_odd++;

    // For even length string
    // no odd freq character
    if (n % 2 == 0) {
        if (count_odd > 0)
            return false;
        else
            return true;
    }

    // For odd length string
    // one odd freq character
    if (count_odd != 1)
        return false;

    return true;
}

// Function for odd freq char and
// reducing its freq by 1, returns
// garbage value if odd freq
// char is not present
static char findOddAndRemoveItsFreq
                    (int freq[])
{
    char odd_char = 'a';

    for (int i = 0; i < MAX_CHAR; i++)
    {
        if (freq[i] % 2 != 0) {
            freq[i]--;
            odd_char = (char)(i + 'a');
            break;
        }
    }

    return odd_char;
}

// To find lexicographically
// first palindromic string.
static boolean findPalindromicString
            (char[] str, int n)
{
    int[] freq = new int[MAX_CHAR] ;
    countFreq(str, freq, n);

    // check whether a palindromic
    // string can be formed or not
    // with the characters of 'str'
    if (!canMakePalindrome(freq, n))
        return false;

    // Assigning odd freq character if
    // present else some garbage value
    char odd_char =
        findOddAndRemoveItsFreq(freq);

    // indexes to store characters
    // from the front and end
    int front_index = 0,
        rear_index = n - 1;

    // Traverse characters
    // in alphabetical order
    for (int i = 0; i < MAX_CHAR; i++)
    {
        if (freq[i] != 0) {
            char ch = (char)(i + 'a');

            // store 'ch' to half its frequency
            // times from the front and rear
            // end. Note that odd character is
            // removed by findOddAndRemoveItsFreq()
            for (int j = 1; j <= freq[i] / 2; j++){
                str[front_index++] = ch;
                str[rear_index--] = ch;
            }
        }
    }

    // if true then odd frequency char exists
    // store it to its corresponding index
    if (front_index == rear_index)
        str[front_index] = odd_char;

    // palindromic string can be formed
    return true;
}

// function to reverse the characters
// in the range i to j in 'str'
static void reverse(char[] str, int i, int j)
{
    char temp;
    while (i < j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;

        i++;
        j--;
    }
}

// function to find next higher
// palindromic string using the
// same set of characters
static Boolean nextPalin(char[] str, int n)
{
    // if length of 'str' is less
    // than '3' then no higher
    // palindromic string can be formed
    if (n <= 3)
        return false;

    // find the index of last character
    // in the 1st half of 'str'
    int mid = n / 2 - 1;
    int i, j;
    // Start from the (mid-1)th character
    // and find the the first character
    // that is alphabetically smaller
    // than the character next to it.
    for (i = mid - 1; i >= 0; i--)
        if (str[i] < str[i + 1])
            break;

    // If no such character is found,
    // then all characters are in
    // descending order (alphabetcally)
    // which means there cannot be a
    // higher palindromic string
    // with same set of characters
    if (i < 0)
        return false;

    // Find the alphabetically smallest
    // character on right side of ith
    // character which is greater than
    // str[i] up to index 'mid'
    int smallest = i + 1;
    for (j = i + 2; j <= mid; j++)
        if (str[j] > str[i] && str[j]
                     < str[smallest])
            smallest = j;

char temp;

    // swap str[i] with str[smallest]
    temp = str[i];
    str[i] = str[smallest];
    str[smallest] = temp;

    // as the string is a palindrome,
    // the same swap of characters
    // should be performed in the
    // 2nd half of 'str'
    temp = str[n - i - 1];
    str[n - i - 1] = str[n - smallest - 1];
    str[n - smallest - 1] = temp;

    // reverse characters in the
    // range (i+1) to mid
    reverse(str, i + 1, mid);

    // if n is even, then reverse
    // characters in the range
    // mid+1 to n-i-2
    if (n % 2 == 0)
        reverse(str, mid + 1, n - i - 2);

    // else if n is odd, then
    // reverse characters in
    // the range mid+2 to n-i-2
    else
        reverse(str, mid + 2, n - i - 2);

    // next alphabetically higher
    // palindromic string can be formed
    return true;
}

// function to print all palindromic
// permutations alphabetically
static void printAllPalinPermutations
            (char[] str, int n) {

    // check if lexicographically
    // first palindromic string can
    // be formed or not using the
    // characters of 'str'
    if (!(findPalindromicString(str, n)))
    {
        // cannot be formed
        System.out.print("-1");
        return;
    }

    // print all palindromic permutations
    do {
        System.out.println(str);
    } while (nextPalin(str, n));
}

// Driver program
public static void main(String[] args)
{

    char[] str = ("malayalam").toCharArray();
    int n = str.length;

    printAllPalinPermutations(str, n);
}
}
```

```
// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 implementation to print all the
# palindromic permutations alphabetically

# import library
import numpy as np

MAX_CHAR = 26

# Function to count frequency of each in the
# string. freq[0] for 'a', ...., freq[25] for 'z'
def countFreq(str, freq, n) :
    for i in range(0, n) :
        freq[ord(str[i]) - ord('a')] += 1


# Cases to check whether a palindromic
# string can be formed or not
def canMakePalindrome(freq, n) :

    # count_odd to count no of
    # s with odd frequency
    count_odd = 0
    for i in range(0, MAX_CHAR) :

        if (freq[i] % 2 != 0):
            count_odd += 1

    # For even length string
    # no odd freq character
    if (n % 2 == 0):
        if (count_odd > 0):
            return False
        else:
            return True


    # For odd length string
    # one odd freq character
    if (count_odd != 1):
        return False

    return True
```

```
# Function to find odd freq and reducing
# its freq by 1, returns garbage
# value if odd freq is not present
def findOddAndRemoveItsFreq(freq) :

    odd_char = 0

    for i in range(0, MAX_CHAR) :
        if (freq[i] % 2 != 0):
            freq[i] = freq[i] - 1
            odd_char = (chr)(i + ord('a'))
            break

    return odd_char


# To find lexicographically first
# palindromic string.
def findPalindromicString(str, n) :

    freq = np.zeros(26, dtype = np.int)
    countFreq(str, freq, n)

    # Check whether a palindromic string
    # can be formed or not with the
    # characters of 'str'
    if (not(canMakePalindrome(freq, n))):
        # cannot be formed
        return False

    # Assigning odd freq character if
    # present else some garbage value
    odd_char = findOddAndRemoveItsFreq(freq)

    # indexes to store acters from
    # the front and end
    front_index = 0; rear_index = n - 1

    # Traverse acters in alphabetical order
    for i in range(0, MAX_CHAR) :
        if (freq[i] != 0) :
            ch = (chr)(i + ord('a'))

            # Store 'ch' to half its frequency times
            # from the front and rear end. Note that
            # odd character is removed by
            # findOddAndRemoveItsFreq()
            for j in range(1, int(freq[i]/2) + 1):
                str[front_index] = ch
                front_index += 1

                str[rear_index] = ch
                rear_index -= 1


    # if True then odd frequency exists
    # store it to its corresponding index
    if (front_index == rear_index):
        str[front_index] = odd_char

    # palindromic string can be formed
    return True


# Function to reverse the acters in the
# range i to j in 'str'
def reverse( str, i, j):

    while (i < j):
        str[i], str[j] = str[j], str[i]
        i += 1
        j -= 1



# Function to find next higher palindromic
# string using the same set of acters
def nextPalin(str, n) :

    # If length of 'str' is less than '3'
    # then no higher palindromic string
    # can be formed
    if (n <= 3):
        return False

    # Find the index of last acter
    # in the 1st half of 'str'
    mid = int (n / 2) - 1


    # Start from the (mid-1)th acter and
    # find the the first acter that is
    # alphabetically smaller than the
    # acter next to it
    i = mid -1
    while(i >= 0) :
        if (str[i] < str[i + 1]):
            break
        i -= 1

    # If no such character is found, then
    # all characters are in descending order
    # (alphabetcally) which means there cannot
    # be a higher palindromic string with same
    # set of characters
    if (i < 0):
        return False

    # Find the alphabetically smallest character
    # on right side of ith character which is
    # greater than str[i] up to index 'mid'
    smallest = i + 1;
    for j in range(i + 2, mid + 1):
        if (str[j] > str[i] and str[j] < str[smallest]):
            smallest = j

    # Swap str[i] with str[smallest]
    str[i], str[smallest] = str[smallest], str[i]

    # As the string is a palindrome, the same
    # swap of characters should be performed
    # in the 2nd half of 'str'
    str[n - i - 1], str[n - smallest - 1] = str[n - smallest - 1], str[n - i - 1]

    # Reverse characters in the range (i+1) to mid
    reverse(str, i + 1, mid)

    # If n is even, then reverse characters in the
    # range mid+1 to n-i-2
    if (n % 2 == 0):
        reverse(str, mid + 1, n - i - 2)

    # else if n is odd, then reverse acters
    # in the range mid+2 to n-i-2
    else:
        reverse(str, mid + 2, n - i - 2)

    # Next alphabetically higher palindromic
    # string can be formed
    return True


# Function to prall the palindromic
# permutations alphabetically
def printAllPalinPermutations(str, n) :

    # Check if lexicographically first
    # palindromic string can be formed
    # or not using the acters of 'str'
    if (not(findPalindromicString(str, n))):
           # cannot be formed
        print ("-1")
        return

    # Converting list into string
    print ( "".join(str))

    # print all palindromic permutations
    while (nextPalin(str, n)):

        # converting list into string
        print ("".join(str))


# Driver Code
str= "malayalam"
n = len(str)

# Convertnig string into list so that
# replacement of characters would be easy
str = list(str)

printAllPalinPermutations(str, n)


# This article is contributed by 'saloni1297'
```

C#

```
 // C# code to print all the palindromic
// permutations alphabetically
using System;

class GFG {

static int MAX_CHAR = 26;

// Function to count frequency
// of each char in the string.
// freq[0] for 'a', ...., freq[25] for 'z'
static void countFreq(char []str, int []freq, int n)
```

```
{
    for (int i = 0; i < n; i++)
        freq[str[i] - 'a']++;
}

// Cases to check whether a palindromic
// string can be formed or not
static Boolean canMakePalindrome(int []freq, int n)
{
    // count_odd to count no of
    // chars with odd frequency
    int count_odd = 0;

    for (int i = 0; i < MAX_CHAR; i++)
        if (freq[i] % 2 != 0)
            count_odd++;

    // For even length string
    // no odd freq character
    if (n % 2 == 0) {
        if (count_odd > 0)
            return false;
        else
            return true;
    }

    // For odd length string
    // one odd freq character
    if (count_odd != 1)
        return false;

    return true;
}

// Function for odd freq char and
// reducing its freq by 1, returns
// garbage value if odd freq
// char is not present
static char findOddAndRemoveItsFreq(int []freq)
{
    char odd_char = 'a';

    for (int i = 0; i < MAX_CHAR; i++)
    {
        if (freq[i] % 2 != 0) {
            freq[i]--;
            odd_char = (char)(i + 'a');
            break;
        }
    }

    return odd_char;
}

// To find lexicographically
// first palindromic string.
static Boolean findPalindromicString(char[] str, int n)
{
    int[] freq = new int[MAX_CHAR] ;
    countFreq(str, freq, n);

    // check whether a palindromic
    // string can be formed or not
    // with the characters of 'str'
    if (!canMakePalindrome(freq, n))
        return false;

    // Assigning odd freq character if
    // present else some garbage value
    char odd_char =
        findOddAndRemoveItsFreq(freq);

    // indexes to store characters
    // from the front and end
    int front_index = 0,
        rear_index = n - 1;

    // Traverse characters
    // in alphabetical order
    for (int i = 0; i < MAX_CHAR; i++)
    {
        if (freq[i] != 0) {
            char ch = (char)(i + 'a');

            // store 'ch' to half its frequency
            // times from the front and rear
            // end. Note that odd character is
            // removed by findOddAndRemoveItsFreq()
            for (int j = 1; j <= freq[i] / 2; j++){
                str[front_index++] = ch;
                str[rear_index--] = ch;
            }
        }
    }

    // if true then odd frequency char exists
    // store it to its corresponding index
    if (front_index == rear_index)
        str[front_index] = odd_char;

    // palindromic string can be formed
    return true;
}

// function to reverse the characters
// in the range i to j in 'str'
static void reverse(char[] str, int i, int j)
{
    char temp;
    while (i < j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;

        i++;
        j--;
    }
}

// function to find next higher
// palindromic string using the
// same set of characters
static Boolean nextPalin(char[] str, int n)
{
    // if length of 'str' is less
    // than '3' then no higher
    // palindromic string can be formed
    if (n <= 3)
        return false;

    // find the index of last character
    // in the 1st half of 'str'
    int mid = n / 2 - 1;
    int i, j;

    // Start from the (mid-1)th character
    // and find the the first character
    // that is alphabetically smaller
    // than the character next to it.
    for (i = mid - 1; i >= 0; i--)
        if (str[i] < str[i + 1])
            break;

    // If no such character is found,
    // then all characters are in
    // descending order (alphabetcally)
    // which means there cannot be a
    // higher palindromic string
    // with same set of characters
    if (i < 0)
        return false;

    // Find the alphabetically smallest
    // character on right side of ith
    // character which is greater than
    // str[i] up to index 'mid'
    int smallest = i + 1;
    for (j = i + 2; j <= mid; j++)
        if (str[j] > str[i] && str[j]
                    < str[smallest])
            smallest = j;

char temp;

    // swap str[i] with str[smallest]
    temp = str[i];
    str[i] = str[smallest];
    str[smallest] = temp;

    // as the string is a palindrome,
    // the same swap of characters
    // should be performed in the
    // 2nd half of 'str'
    temp = str[n - i - 1];
    str[n - i - 1] = str[n - smallest - 1];
    str[n - smallest - 1] = temp;

    // reverse characters in the
    // range (i+1) to mid
    reverse(str, i + 1, mid);

    // if n is even, then reverse
    // characters in the range
    // mid+1 to n-i-2
    if (n % 2 == 0)
        reverse(str, mid + 1, n - i - 2);

    // else if n is odd, then
    // reverse characters in
    // the range mid+2 to n-i-2
    else
        reverse(str, mid + 2, n - i - 2);
    // next alphabetically higher
    // palindromic string can be formed
    return true;
}

// function to print all palindromic
// permutations alphabetically
static void printAllPalinPermutations(char[] str, int n)
{
    // check if lexicographically
    // first palindromic string can
    // be formed or not using the
    // characters of 'str'
    if (!(findPalindromicString(str, n)))
    {
        // cannot be formed
        Console.WriteLine("-1");
        return;
    }

    // print all palindromic permutations
    do {
    Console.WriteLine(str);
    } while (nextPalin(str, n));
}

// Driver program
public static void Main(String[] args)
{
    char[] str = ("malayalam").ToCharArray();
    int n = str.Length;

    printAllPalinPermutations(str, n);
}
}

// This code is contributed by parashar.
```

Output :

```
aalmymlaa
aamlylmaa
alamymala
almayamla
amalylama
amlayalma
```

```
laamymaal
lamayamal
lmaayaaml
maalylaam
malayalam
mlaayaalm
```

Time Complexity: O(m*n), where **m** is the number of palindromic permutations of **str**.

Improved By : parashar

Source

https://www.geeksforgeeks.org/print-palindromic-permutations-given-string-alphabetic-order/

Chapter 112

Print distinct sorted permutations with duplicates allowed in input

Print distinct sorted permutations with duplicates allowed in input - GeeksforGeeks

Write a program to print all distinct permutations of a given string in sorted order. Note that the input string may contain duplicate characters.

In mathematics, the notion of permutation relates to the act of arranging all the members of a set into some sequence or order, or if the set is already ordered, rearranging (reordering) its elements, a process called permuting.
Source – Wikipedia

Examples:

> Input : BAC
> Output : ABC ACB BAC BCA CAB CBA
>
> Input : AAB
> Output : AAB ABA BAA
>
> Input : DBCA
> Output: ABCD ABDC ACBD ACDB ADBC ADCB BACD BADC BCAD BCDA BDAC BDCA CABD CADB CBAD CBDA CDAB CDBA DABC DACB DBAC DBCA DCAB DCBA

Concept Used : The number of Strings generated by a string of distinct characters of length 'n' is equal to 'n!'. Sorting any given string and generating the lexicographically next bigger string till we reach the largest lexicographically string from those characters.

> Different permutations of word "geeks"
> Length of string = 5

Character 'e' repeats 2 times.
Result = 5!/2! = 60.

Steps : Example : Consider a string "ABCD".

Step 1 : Sort the string .
Step 2 : Obtain the total number of permutations which can be formed from that string.
Step 3 : Print the sorted string and then loop for number of (permutations-1) times as 1st string is already printed.
Step 4 : Find next greater string,.

Here is the Java Implementation of this problem –

```
 // Java program to print all permutations of a string
// in sorted order.
import java.io.*;
import java.util.*;

class Solution {

  // Calculating factorial of a number
  static int factorial(int n) {
    int f = 1;
    for (int i = 1; i <= n; i++)
      f = f * i;
    return f;
  }

  // Method to print the array
  static void print(char[] temp) {
    for (int i = 0; i < temp.length; i++)
      System.out.print(temp[i]);
    System.out.println();
  }

  // Method to find total number of permutations
  static int calculateTotal(char[] temp, int n) {
    int f = factorial(n);

    // Building HashMap to store frequencies of
    // all characters.
    HashMap<Character, Integer> hm =
                     new HashMap<Character, Integer>();
    for (int i = 0; i < temp.length; i++) {
      if (hm.containsKey(temp[i]))
        hm.put(temp[i], hm.get(temp[i]) + 1);
      else
        hm.put(temp[i], 1);
    }
    // Traversing hashmap and finding duplicate elements.
    for (Map.Entry e : hm.entrySet()) {
      Integer x = (Integer)e.getValue();
      if (x > 1) {
        int temp5 = factorial(x);
        f = f / temp5;
      }
    }
    return f;
  }

  static void nextPermutation(char[] temp) {

    // Start traversing from the end and
    // find position 'i-1' of the first character
    // which is greater than its  successor.
    int i;
    for (i = temp.length - 1; i > 0; i--)
      if (temp[i] > temp[i - 1])
        break;

    // Finding smallest character after 'i-1' and
    // greater than temp[i-1]
    int min = i;
    int j, x = temp[i - 1];
    for (j = i + 1; j < temp.length; j++)
      if ((temp[j] < temp[min]) && (temp[j] > x))
        min = j;

    // Swapping the above found characters.
    char temp_to_swap;
    temp_to_swap = temp[i - 1];
    temp[i - 1] = temp[min];
    temp[min] = temp_to_swap;

    // Sort all digits from position next to 'i-1'
    // to end of the string.
    Arrays.sort(temp, i, temp.length);

    // Print the String
    print(temp);
  }

  static void printAllPermutations(String s) {

    // Sorting String
    char temp[] = s.toCharArray();
    Arrays.sort(temp);

    // Print first permutation
    print(temp);

    // Finding the total permutations
    int total = calculateTotal(temp, temp.length);
    for (int i = 1; i < total; i++)
      nextPermutation(temp);
  }

  // Driver Code
  public static void main(String[] args) {
    String s = "AAB";
    printAllPermutations(s);
  }
}
```

Output:

```
AAB
ABA
BAA
```

Time Complexity: O(n*m) where n is size of array and m is number of permutations possible
.
Auxiliary Space: O(n).

Source

https://www.geeksforgeeks.org/print-distinct-sorted-permutations-with-duplicates-allowed/

Chapter 113

Probability for three randomly chosen numbers to be in AP

Probability for three randomly chosen numbers to be in AP - GeeksforGeeks

Given a number **n** and an array containing 1 to (2n+1) consecutive numbers. Three elements are chosen at random. Find the probability that the elements chosen are in A.P.

Examples:

```
Input : n = 2
Output : 0.4
The array would be {1, 2, 3, 4, 5}
Out of all elements, triplets which
are in AP: {1, 2, 3}, {2, 3, 4},
{3, 4, 5}, {1, 3, 5}
No of ways to choose elements from
the array: 10 (5C3)
So, probability = 4/10 = 0.4

Input : n = 5
Output : 0.1515
```

The number of ways to select any 3 numbers from (2n+1) numbers are:$^{\mathbf{(2n\,+\,1)}}\,\mathbf{C}_{\mathbf{3}}$
Now, for the numbers to be in AP:
with common difference 1—{1, 2, 3}, {2, 3, 4}, {3, 4, 5}…{2n-1, 2n, 2n+1}
with common difference 2—{1, 3, 5}, {2, 4, 6}, {3, 5, 7}…{2n-3, 2n-1, 2n+1}
with common difference n— {1, n+1, 2n+1}
Therefore, Total number of AP group of 3 numbers in (2n+1) numbers are:
$(2n - 1)+(2n - 3)+(2n - 5) +\dots+ 3 + 1 =$ **n * n** (Sum of first n odd numbers is n * n)
So, probability for 3 randomly chosen numbers in (2n + 1) consecutive numbers to be in AP = (n * n) / $^{(2n\,+\,1)}\,C_{3}$ = **3 n / (4 (n * n) – 1)**

C++

```
 // CPP program to find probability that
// 3 randomly chosen numbers form AP.
#include <bits/stdc++.h>
using namespace std;

// function to calculate probability
double procal(int n)
{
    return (3.0 * n) / (4.0 * (n * n) - 1);
}

// Driver code to run above function
int main()
{
    int a[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(a)/sizeof(a[0]);
    cout << procal(n);
    return 0;
}
```

Java

```
 // Java program to find probability that
// 3 randomly chosen numbers form AP.

class GFG {

    // function to calculate probability
    static double procal(int n)
    {
        return (3.0 * n) / (4.0 * (n * n) - 1);
    }

    // Driver code to run above function
    public static void main(String arg[])
    {
        int a[] = { 1, 2, 3, 4, 5 };
        int n = a.length;
        System.out.print(Math.round(procal(n) * 1000000.0) / 1000000.0);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find probability that
# 3 randomly chosen numbers form AP.

# Function to calculate probability
def procal(n):

    return (3.0 * n) / (4.0 * (n * n) - 1)

# Driver code
a = [1, 2, 3, 4, 5]
n = len(a)
print(round(procal(n), 6))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find probability that
// 3 randomly chosen numbers form AP.
using System;

class GFG {

    // function to calculate probability
    static double procal(int n)
    {
        return (3.0 * n) / (4.0 * (n * n) - 1);
    }

    // Driver code
    public static void Main()
    {
        int []a = { 1, 2, 3, 4, 5 };
        int n = a.Length;
        Console.Write(Math.Round(procal(n) *
                    1000000.0) / 1000000.0);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find probability that
// 3 randomly chosen numbers form AP.
// function to calculate probability
function procal($n)
{
    return (3.0 * $n) /
           (4.0 * ($n *
              $n) - 1);
}

    // Driver code
    $a = array(1, 2, 3, 4, 5);
    $n = sizeof($a);
    echo procal($n);

// This code is contributed by aj_36
?>
```

Output:

```
0.151515
```

Time Complexity : O(1)

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/probability-three-randomly-chosen-numbers-ap/

Chapter 114

Probability that the pieces of a broken stick form a n sided polygon

Probability that the pieces of a broken stick form a n sided polygon - GeeksforGeeks

We have a stick of length L. The stick got broken at (n-1) randomly chosen points (lengths of parts can be non-integer or floating point numbers also) so we get n parts. We need to find the probability that these n pieces can form a n sided polygon.

Examples:

```
Input  : L = 5 n = 3
Output : 0.25
We need to cut rope of length 5
into three parts.
```

First we need to find the condition when n lengths can form a n sided polygon. Let us consider a triangle, we know for a triangle the length of the largest side must be smaller than the sum of the lengths of other sides. Similarly for a n sided polygon the the length of the largest side must be less than the sum of the other (n-1) sides. Why? Suppose we break the stick into two equal halves. We further break one of the halves into (n-1) parts. We can never place them such that they make a closed polygon. (Actually the best we can do is to make 2 parallel lines). So we just need to find the probability that no part has the length greater than or equal to L/2.

Now we need to work on the probability. There are many ways to calculate the required probability we will use a geometric approach. Consider a circle of perimeter L. We place n points on the perimeter. The probability that they lie on the same semicircle is $\frac{n}{2^{n-1}}$. Please refer this link for more information, let us denote it by P(E).

This probability is actually same as breaking the stick such that at least one part is of length L/2. But we want just the complement of this event hence our answer is [illegible]

C++

```
 // CPP program to find probability that
// a rope of length L when broken into
// n parts form a polygon.
#include<iostream>

using namespace std;

double printProbability(unsigned L, unsigned n)
{
   unsigned p = (1 << (n-1));
   return 1.0 - ((double)n) / ((double)p);
}

int main()
{
   unsigned n = 3, L = 5;
   cout << printProbability(L, n);
   return 0;
}
```

PHP

Output:

```
0.25
```

Improved By : SujanDutta, Abby_akku

Source

https://www.geeksforgeeks.org/probability-that-the-pieces-of-a-broken-stick-form-a-n-sided-polygon/

Chapter 115

Problem on permutations and combinations | Set 2

Problem on permutations and combinations | Set 2 - GeeksforGeeks

Prerequisite : Permutation and Combination

Given a polygon of m sides, count number of triangles that can be formed using vertices of polygon.

> Answer : [m (m – 1)(m – 2) / 6]
> Explanation : There are m vertices in a polygon with m sides. We need to count different combinations of three points chosen from m. So answer is ${}^{m}C_3$ = m * (m – 1) * (m – 2) / 6
> Examples :
> Input: m = 3
> Output: 1
> We put value of m = 3, we get required no. of triangles = 3 * 2 * 1 / 6 = 1
>
> Input : m = 6
> output : 20

Given a polygon of m sides, count number of diagonals that can be formed using vertices of polygon.

> Answer : [m (m – 3)] / 2.
> Explanation : We need to choose two vertices from polygon. We can choose first vertex m ways. We can choose second vertex in m-3 ways (Note that we can not choose adjacent two vertices to form a diagonal). So total number is m * (m – 3). This is twice the total number of combinations as we consider an diagonal edge u-v twice (u-v and v-u)
> Examples:
> Input m = 4

output: 2
We put the value of m = 4, we get the number of required diagonals = 4 * (4 – 3) / 2 = 2

Input: m = 5
Output: 5

Count the total number rectangles that can be formed using m vertical lines and n horizontal lines

Answer : $({}^mC_2 * {}^nC_2)$.
We need to choose two vertical lines and two horizontal lines. Since the vertical and horizontal lines are chosen independently, we multiply the result.
Examples:
Input: m = 2, n = 2
Output: 1
We have the total no of rectangles
$= {}^2C_2 * {}^2C_2$
= 1 * 1 = 1

Input: m = 4, n = 4
Output: 36

There are 'n' points in a plane, out of which 'm' points are co-linear. Find the number of triangles formed by the points as vertices ?

Number of triangles = $\mathbf{{}^nC_3 - {}^mC_3}$
Explanation : Consider the example n = 10, m = 4. There are 10 points, out of which 4 collinear. A triangle will be formed by any three of these ten points. Thus forming a triangle amounts to selecting any three of the 10 points. Three points can be selected out of the 10 points in ${}^nC_3$ ways.
Number of triangles formed by 10 points when no 3 of them are co-linear = ${}^{10}C_3$……(i)
Similarly, the number of triangles formed by 4 points when no 3 of them are co-linear = ${}^4C_3$……..(ii)

Since triangle formed by these 4 points are not valid, required number of triangles formed = ${}^{10}C_3 - {}^4C_3$ = 120 – 4 = 116

There are 'n' points in a plane out of which 'm' points are collinear, count the number of distinct straight lines formed by joining two points.

Answer : ${}^nC_2 - {}^mC_2 + 1$
Explanation : Number of straight lines formed by n points when none of them are col-linear = ${}^nC_2$
Similarly, the number of straight lines formed by m points when none of them collinear = ${}^mC_2$
m points are collinear and reduce in one line. Therefore we subtract ${}^mC_2$ and

add 1.
Hence answer = $^nC_2$ – $^mC_2$ +1
Examples:
Input : n = 4, m = 3
output : 1
We apply this formula
Answer = $^4C_2$ – $^3C_2$ +1
= 3 – 3 + 1
= 1

More practice questions on permutation and combination :
Quiz on Permutation and Combination
Combination and Permutation Practice Questions

Source

https://www.geeksforgeeks.org/problem-permutations-combinations-set-2/

Chapter 116

Program for Binomial Coefficients table

Program for Binomial Coefficients table - GeeksforGeeks

Given an integer max, print Binomial Coefficientstable that prints all binomial coefficients B(m, x) where m and x vary from 0 to max

Example :

```
Input : max = 3
Output :
 0   1
 1   1   1
 2   1   2   1
 3   1   3   3   1
```

The easiest way to explain what binomial coefficients are is to say that they count certain ways of grouping items. Specifically, the binomial coefficient B(m, x) counts the number of ways to form an unordered collection of k items chosen from a collection of n distinct items. Binomial coefficients are used in the study of binomial distributions and multicomponent redundant systems. It is given by [illegible]

Example :

```
Compute B(7, 3)  where m = 7 and x = 1
          (7!/3!(7-3)!)7
        = 7!/3!*4!
        = (7*6*5*4*3*2*1)/(3*2*1)*(4*3*2*1)
        = 35
```

A table of binomial coefficients is required to determine the binomial coefficient for any value m and x.

Problem Analysis :
The binomial coefficient can be recursively calculated as follows –
[illegible] further, [illegible]

That is the binomial coefficient is one when either x is zero or m is zero. The program prints the table of binomial coefficients for [illegible].

C++

```
 // C++ program for binomial coefficients
#include <stdio.h>

// Function to print binomial table
int printbinomial(int max)
{
    for (int m = 0; m <= max; m++) {
        printf("%2d", m);
        int binom = 1;
        for (int x = 0; x <= m; x++) {

            // B(m, x) is 1 if either m or x is
            // is 0.
            if (m != 0 && x != 0)

                // Otherwise using recursive formula
                // B(m, x) = B(m, x - 1) * (m - x + 1) / x
                binom = binom * (m - x + 1) / x;

            printf("%4d", binom);
        }
        printf("\n");
    }
}

// Driver Function
int main()
{
    int max = 10;
    printbinomial(max);
    return 0;
}
```

Java

```
 // Java program for
// binomial coefficients
import java.io.*;

class GFG
{

// Function to print
// binomial table
static void printbinomial(int max)
{
    for (int m = 0; m <= max; m++)
    {
        System.out.print(m + " ");
        int binom = 1;
        for (int x = 0; x <= m; x++)
        {

            // B(m, x) is 1 if either
            // m or x is is 0.
            if (m != 0 && x != 0)

                // Otherwise using
                // recursive formula
                // B(m, x) = B(m, x - 1) *
                //              (m - x + 1) / x
                binom = binom * (m - x + 1) / x;

            System.out.print(binom + " ");
        }
        System.out.println();
    }
}

// Driver Code
public static void main (String[] args)
{
    int max = 10;
    printbinomial(max);
}
}

// This code is contributed
// by akt_mit
```

Python3

```
 # Python3 program for binomial
```

```
# coefficients

# Function to print binomial table
def printbinomial (max):

    for m in range(max + 1):
        print( '% 2d'% m, end = ' ')
        binom = 1
        for x in range(m + 1):

            # B(m, x) is 1 if either m
            # or x is is 0.
            if m != 0 and x != 0:

                # Otherwise using recursive
                # formula
                # B(m, x) = B(m, x - 1) *
                #           (m - x + 1) / x
                binom = binom * (m - x + 1) / x
            print( '% 4d'% binom, end = ' ')
        print("\n", end = '')

# Driver Function
max = 10
printbinomial(max)

# This code is contributed by "Sharad_bhardwaj".
```

C#

```
 // C# program for binomial coefficients
using System;

public class GFG {

    // Function to print binomial table
    static void printbinomial(int max)
    {
        for (int m = 0; m <= max; m++) {
            Console.Write(m + " ");
            int binom = 1;
            for (int x = 0; x <= m; x++) {

                // B(m, x) is 1 if either m
                // or x is is 0.
                if (m != 0 && x != 0)

                    // Otherwise using recursive formula
                    // B(m, x) = B(m, x - 1) * (m - x + 1) / x
                    binom = binom * (m - x + 1) / x;

                Console.Write(binom + " ");
            }
            Console.WriteLine();
        }
    }

    // Driver Function
    static public void Main()
    {
        int max = 10;
        printbinomial(max);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for
// binomial coefficients

// Function to print
// binomial table
function printbinomial($max)
{
    for ($m = 0; $m <= $max; $m++)
    {
        echo $m;
        $binom = 1;
        for ($x = 0; $x <= $m; $x++)
        {

            // B(m, x) is 1 if either
            // m or x is 0.
            if ($m != 0 && $x != 0)

                // Otherwise using
                // recursive formula
                // B(m, x) = B(m, x - 1) *
                //            (m - x + 1) / x
                $binom = $binom * ($m - $x + 1) / $x;

            echo " ", $binom, " ";
        }
    echo "\n";
    }
}

// Driver Code
$max = 10;
printbinomial($max);

// This code is contributed by aj_36
?>
```

Output :

```
 0   1
 1   1   1
 2   1   2   1
 3   1   3   3   1
 4   1   4   6   4   1
 5   1   5  10  10   5   1
 6   1   6  15  20  15   6   1
 7   1   7  21  35  35  21   7   1
 8   1   8  28  56  70  56  28   8   1
 9   1   9  36  84 126 126  84  36   9   1
10   1  10  45 120 210 252 210 120  45  10   1
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/program-binomial-coefficients-table/

Chapter 117

Program to calculate value of nCr

Program to calculate value of nCr - GeeksforGeeks

Following are common definition of Binomial Coefficients.

1. A binomial coefficient C(n, k) can be defined as the coefficient of X^k in the expansion of (1 + X)^n.
2. A binomial coefficient C(n, k) also gives the number of ways, disregarding order, that k objects can be chosen from among n objects; more formally, the number of k-element subsets (or k-combinations) of an n-element set.

Given two numbers n and r, find value of ${}^{n}C_{r}$

Examples :

```
Input :  n = 5, r = 2
Output : 30
The value of 5C2 is 10

Input : n = 3, r = 1
Output : 3
```

The idea is simply based on below formula.

${}^{n}C_{r}$ = (n!) / (r! * (n-r)!)

C++

```
 // CPP program To calculate The Value Of nCr
#include <bits/stdc++.h>
using namespace std;

int fact(int n);

int nCr(int n, int r)
{
    return fact(n) / (fact(r) * fact(n - r));
}

// Returns factorial of n
int fact(int n)
{
    int res = 1;
    for (int i = 2; i <= n; i++)
        res = res * i;
    return res;
}

// Driver code
int main()
{
    int n = 5, r = 3;
    cout << nCr(n, r);
    return 0;
}
```

Java

```
 // Java program To calculate
// The Value Of nCr
class GFG {

static int nCr(int n, int r)
{
    return fact(n) / (fact(r) *
                  fact(n - r));
}

// Returns factorial of n
static int fact(int n)
{
    int res = 1;
    for (int i = 2; i <= n; i++)
        res = res * i;
    return res;
}
// Driver code
public static void main(String[] args)
{
    int n = 5, r = 3;
    System.out.println(nCr(n, r));
}
}

// This code is Contributed by
// Smitha Dinesh Semwal.
```

Python 3

```
 # Python 3 program To calculate
# The Value Of nCr

def nCr(n, r):

    return (fact(n) / (fact(r)
                * fact(n - r)))

# Returns factorial of n
def fact(n):

    res = 1

    for i in range(2, n+1):
        res = res * i

    return res

# Driver code
n = 5
r = 3
print(int(nCr(n, r)))

# This code is contributed
# by Smitha
```

C#

```
 // C# program To calculate
// The Value Of nCr
using System;

class GFG {
```

```
static int nCr(int n, int r)
{
   return fact(n) / (fact(r) *
                 fact(n - r));
}

// Returns factorial of n
static int fact(int n)
{
    int res = 1;
    for (int i = 2; i <= n; i++)
        res = res * i;
    return res;
}

   // Driver code
   public static void Main()
   {
      int n = 5, r = 3;
      Console.Write(nCr(n, r));
   }
}

// This code is Contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program To calculate
// the Value Of nCr


function nCr( $n, $r)
{
    return fact($n) / (fact($r) *
                  fact($n - $r));
}

// Returns factorial of n
function fact( $n)
{
    $res = 1;
    for ( $i = 2; $i <= $n; $i++)
        $res = $res * $i;
    return $res;
}
```

```
    // Driver code
    $n = 5;
    $r = 3;
    echo nCr($n, $r);

// This code is contributed by vt_m.
?>
```

Output:

```
10
```

More Efficient Solutions:
Dynamic Programming | Set 9 (Binomial Coefficient)
Space and time efficient Binomial Coefficient

All Articles on Binomial Coefficient

Improved By : Smitha Dinesh Semwal, nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/program-calculate-value-ncr/

Chapter 118

Program to print binomial expansion series

Program to print binomial expansion series - GeeksforGeeks

Given three integers, A, X and n, the task is to print terms of below binomial expression series.

$$\mathbf{(A+X)^n = {}^nC_0A^nX^0 + {}^nC_1A^{n\text{-}1}X^1 + {}^nC_2A^{n\text{-}2}X^2 +....+ {}^nC_nA^0X^n}$$

Examples:

```
Input : A = 1, X = 1, n = 5
Output : 1 5 10 10 5 1

Input : A = 1, B = 2, n = 6
Output : 1 12 60 160 240 192 64
```

Simple Solution : We know that for each value of n there will be (n+1) term in the binomial series. So now we use a simple approach and calculate the value of each element of the series and print it .

```
nCr = (n!) / ((n-r)! * (r)!)

Below is value of general term.
Tr+1 = nCn-rAn-rXr
So at each position we have to find the value
of the general term and print that term .
```

C++

```
 // CPP program to print terms of binomial
// series and also calculate sum of series.
#include <bits/stdc++.h>
using namespace std;

// function to calculate factorial of
// a number
int factorial(int n)
{
    int f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    return f;
}

// fuction to print the series
void series(int A, int X, int n)
{
    // calculating the value of n!
    int nFact = factorial(n);

    // loop to display the series
    for (int i = 0; i < n + 1; i++) {

        // For calculating the
        // value of nCr
        int niFact = factorial(n - i);
        int iFact = factorial(i);

        // calculating the value of
        // A to the power k and X to
        // the power k
        int aPow = pow(A, n - i);
        int xPow = pow(X, i);

        // display the series
        cout << (nFact * aPow * xPow) /
                 (niFact * iFact) << " ";
    }
}

// main fuction started
int main()
{
    int A = 3, X = 4, n = 5;
    series(A, X, n);
    return 0;
}
```

Java

```
 // Java program to print terms of binomial
// series and also calculate sum of series.

import java.io.*;

class GFG {

    // function to calculate factorial of
    // a number
    static int factorial(int n)
    {
        int f = 1;
        for (int i = 2; i <= n; i++)
            f *= i;

        return f;
    }

    // fuction to print the series
    static void series(int A, int X, int n)
    {

        // calculating the value of n!
        int nFact = factorial(n);

        // loop to display the series
        for (int i = 0; i < n + 1; i++) {

            // For calculating the
            // value of nCr
            int niFact = factorial(n - i);
            int iFact = factorial(i);

            // calculating the value of
            // A to the power k and X to
            // the power k
            int aPow = (int)Math.pow(A, n - i);
            int xPow = (int)Math.pow(X, i);

            // display the series
            System.out.print((nFact * aPow * xPow)
                             / (niFact * iFact) + " ");
        }
    }

    // main fuction started
    public static void main(String[] args)
    {
        int A = 3, X = 4, n = 5;

        series(A, X, n);
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python3 program to print terms of binomial
# series and also calculate sum of series.

# function to calculate factorial
# of a number
def factorial(n):

    f = 1
    for i in range(2, n+1):
        f *= i

    return f

# Fuction to print the series
def series(A, X, n):

    # calculating the value of n!
    nFact = factorial(n)

    # loop to display the series
    for i in range(0, n + 1):

        # For calculating the
        # value of nCr
        niFact = factorial(n - i)
        iFact = factorial(i)

        # calculating the value of
        # A to the power k and X to
        # the power k
        aPow = pow(A, n - i)
        xPow = pow(X, i)
        # display the series
        print (int((nFact * aPow * xPow) /
                   (niFact * iFact)), end = " ")

# Driver Code
A = 3; X = 4; n = 5
series(A, X, n)

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to print terms of binomial
// series and also calculate sum of series.
using System;

class GFG {

    // function to calculate factorial of
    // a number
    static int factorial(int n)
    {
        int f = 1;
        for (int i = 2; i <= n; i++)
            f *= i;

        return f;
    }

    // fuction to print the series
    static void series(int A, int X, int n)
    {

        // calculating the value of n!
        int nFact = factorial(n);

        // loop to display the series
        for (int i = 0; i < n + 1; i++) {

            // For calculating the
            // value of nCr
            int niFact = factorial(n - i);
            int iFact = factorial(i);

            // calculating the value of
            // A to the power k and X to
            // the power k
            int aPow = (int)Math.Pow(A, n - i);
            int xPow = (int)Math.Pow(X, i);

            // display the series
            Console.Write((nFact * aPow * xPow)
                     / (niFact * iFact) + " ");
        }
    }

    // main fuction started
    public static void Main()
    {
        int A = 3, X = 4, n = 5;

        series(A, X, n);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to print
// terms of binomial
// series and also
// calculate sum of series.

// function to calculate
// factorial of a number
function factorial($n)
{
    $f = 1;
    for ($i = 2; $i <= $n; $i++)
        $f *= $i;
    return $f;
}

// fuction to print the series
function series($A, $X, $n)
{

    // calculating the
    // value of n!
    $nFact = factorial($n);

    // loop to display
    // the series
    for ($i = 0; $i < $n + 1; $i++)
    {

        // For calculating the
        // value of nCr
        $niFact = factorial($n - $i);
        $iFact = factorial($i);

        // calculating the value of
        // A to the power k and X to
        // the power k
        $aPow = pow($A, $n - $i);
        $xPow = pow($X, $i);

        // display the series
        echo ($nFact * $aPow * $xPow) /
             ($niFact * $iFact) , " ";
    }
}

    // Driver Code
    $A = 3;
    $X = 4;
    $n = 5;
    series($A, $X, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
243 1620 4320 5760 3840 1024
```

Efficient Solution :
The idea is to compute next term using previous term. We can compute next term in O(1) time. We use below property of Binomial Coefficients.

$^nC_{i+1} = {}^nC_i*(n-i)/(i+1)$

C++

```
 // CPP program to print terms of binomial
// series and also calculate sum of series.
#include <bits/stdc++.h>
using namespace std;

// fuction to print the series
void series(int A, int X, int n)
```

```
{
    // Calculating and printing first term
    int term = pow(A, n);
    cout << term << " ";

    // Computing and printing remaining terms
    for (int i = 1; i <= n; i++) {

        // Find current term using previous terms
        // We increment power of X by 1, decrement
        // power of A by 1 and compute nCi using
        // previous term by multiplying previous
        // term with (n - i + 1)/i
        term = term * X * (n - i + 1)/(i * A);

        cout << term << " ";
    }
}

// main fuction started
int main()
{
    int A = 3, X = 4, n = 5;
    series(A, X, n);
    return 0;
}
```

Java

```
 // Java program to print terms of binomial
// series and also calculate sum of series.

import java.io.*;

class GFG {

    // fuction to print the series
    static void series(int A, int X, int n)
    {

        // Calculating and printing first
        // term
        int term = (int)Math.pow(A, n);
        System.out.print(term + " ");

        // Computing and printing
        // remaining terms
        for (int i = 1; i <= n; i++) {
            // Find current term using
            // previous terms We increment
            // power of X by 1, decrement
            // power of A by 1 and compute
            // nCi using previous term by
            // multiplying previous term
            // with (n - i + 1)/i
            term = term * X * (n - i + 1)
                                / (i * A);

            System.out.print(term + " ");
        }
    }

    // main fuction started
    public static void main(String[] args)
    {
        int A = 3, X = 4, n = 5;

        series(A, X, n);
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python 3 program to print terms of binomial
# series and also calculate sum of series.

# Fuction to print the series
def series(A, X, n):

    # Calculating and printing first term
    term = pow(A, n)
    print(term, end = " ")

    # Computing and printing remaining terms
    for i in range(1, n+1):

        # Find current term using previous terms
        # We increment power of X by 1, decrement
        # power of A by 1 and compute nCi using
        # previous term by multiplying previous
        # term with (n - i + 1)/i
        term = int(term * X * (n - i + 1)/(i * A))
        print(term, end = " ")

# Driver Code
A = 3; X = 4; n = 5
series(A, X, n)

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to print terms of binomial
// series and also calculate sum of series.

using System;

public class GFG {

    // fuction to print the series
    static void series(int A, int X, int n)
    {

        // Calculating and printing first
        // term
        int term = (int)Math.Pow(A, n);
        Console.Write(term + " ");

        // Computing and printing
        // remaining terms
        for (int i = 1; i <= n; i++) {

            // Find current term using
            // previous terms We increment
            // power of X by 1, decrement
            // power of A by 1 and compute
            // nCi using previous term by
            // multiplying previous term
            // with (n - i + 1)/i
            term = term * X * (n - i + 1)
                                / (i * A);

          Console.Write(term + " ");
        }
    }

    // main fuction started
    public static void Main()
    {
        int A = 3, X = 4, n = 5;
        series(A, X, n);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to print
// terms of binomial
// series and also
// calculate sum of
// series.

// fuction to print
// the series
function series($A, $X, $n)
{

    // Calculating and printing
    // first term
    $term = pow($A, $n);
    echo $term , " ";

    // Computing and printing
    // remaining terms
    for ($i = 1; $i <= $n; $i++)
    {

        // Find current term
        // using previous terms
        // We increment power
        // of X by 1, decrement
        // power of A by 1 and
        // compute nCi using
        // previous term by
        // multiplying previous
        // term with (n - i + 1)/i
        $term = $term * $X * ($n - $i + 1) /
                             ($i * $A);

        echo $term , " ";
    }
}

    // Driver Code
    $A = 3;
    $X = 4;
    $n = 5;
    series($A, $X, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
243 1620 4320 5760 3840 1024
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/program-print-binomial-expansion-series/

Chapter 119

Rearrange first N numbers to make them at K distance

Rearrange first N numbers to make them at K distance - GeeksforGeeks

Given a positive number K, we need to permute first N natural numbers in such a way that absolute distance of each permuted number from its original position is K and if it is not possible to rearrange them in such manner then print not possible.
Examples :

```
Input  :  N = 12
          K = 2
Output : [3 4 1 2 7 8 5 6 11 12 9 10]
Explanation : Initial permutation is
[1 2 3 4 5 6 7 8 9 10 11 12]
In rearrangement, [3 4 1 2 7 8 5 6 11
12 9 10] we have all elements at
distance 2.

Input  :  N = 12
          K = 3
Output : [4 5 6 1 2 3 10 11 12 7 8 9]
```

We can solve this problem by finding the pattern in solutions. If we go through many solutions manually, we can see that if we partition N numbers into slots of size 2K, then each slot can be rearranged in two parts of size K, where the difference of position with actual position will be K.

```
Example 1 : N = 12 and K = 2
```

```
First 12 numbers are partitioned into
2*2 = 4 sized 12/4 = 3 slots as shown below,
[[1 2 3 4] [5 6 7 8] [9 10 11 12]]

Now each half of the slot is swapped so that,
every number will go at K position distance
from its initial position as shown below,
[[3 4 1 2] [7 8 5 6] [11 12 9 10]]

Example 2 :  N = 12 and K = 3,
[1 2 3 4 5 6 7 8 9 10 11 12]
[[1 2 3 4 5 6] [7 8 9 10 11 12]]
[[4 5 6 1 2 3] [10 11 12 7 8 9]]
[4 5 6 1 2 3 10 11 12 7 8 9]
Which is the final rearrangement for
N = 12 and K = 3
```

In below code, the case when K = 0 is handled separately by printing all numbers in their actual order. When N is not divisible by 2K, 'not possible' is printed directly.

C++

```
 // C++ program to rearrange permuatations to make
// them K distance away
#include <bits/stdc++.h>
using namespace std;

/* Method prints permutation of first N numbers,
where each number is K distance away from its
actual position */
void printRearrangeNnumberForKDistance(int N, int K)
{
    // If K = 0, then print numbers in their natural
    // order
    if (K == 0)
    {
        for (int i = 1; i <= N; i++)
            cout << i << " ";
        cout << endl;
        return;
    }

    // If N doesn't divide (2K) evenly, then
    // rearrangement is not possible
    if (N % (2 * K) != 0)
    {
        cout << "Not Possible\n";
        return;
    }

    // Copy first N numbers to an auxiliary
    // array
    int arr[N + 1];
    for (int i = 1; i <= N; i++)
        arr[i] = i;

    // Swap halves of each 2K sized slot
    for (int i = 1; i <= N; i += 2 * K)
        for (int j = 1; j <= K; j++)
            swap(arr[i + j - 1], arr[K + i + j - 1]);

    // Print the rearranged array
    for (int i = 1; i <= N; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int N = 12, K = 3;
    printRearrangeNnumberForKDistance(N, K);
    return 0;
}
```

Java

```
 // Java program to rearrange permuatations
// to make them K distance away

class GFG
{
    /* Method prints permutation of first N numbers,
    where each number is K distance away from its
    actual position */
    static void printRearrangeNnumberForKDistance(int N, int K)
    {
        // If K = 0, then print numbers
        // in their natural order
        if (K == 0)
        {
            for (int i = 1; i <= N; i++)
                System.out.print(i + " ");
            System.out.println();
            return;
        }
        // If N doesn't divide (2K) evenly, then
        // rearrangement is not possible
        if (N % (2 * K) != 0)
        {
            System.out.print("Not Possible\n");
            return;
        }

        // Copy first N numbers to an auxiliary
        // array
        int arr[]=new int[N + 1];
        for (int i = 1; i <= N; i++)
            arr[i] = i;

        // Swap halves of each 2K sized slot
        for (int i = 1; i <= N; i += 2 * K)
            for (int j = 1; j <= K; j++)
            {
                int temp = arr[i + j - 1];
                arr[i + j - 1] = arr[K + i + j - 1];
                arr[K + i + j - 1] = temp;
            }

        // Print the rearranged array
        for (int i = 1; i <= N; i++)
            System.out.print(arr[i] + " ");
    }

    // Driver code
    public static void main (String[] args)
    {
        int N = 12, K = 3;
        printRearrangeNnumberForKDistance(N, K);
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to rearrange permuatations
// to make them K distance away
using System;

class GFG
{
    /* Method prints permutation of first N numbers,
    where each number is K distance away from its
    actual position */
    static void printRearrangeNnumberForKDistance(int N, int K)
    {
        // If K = 0, then print numbers
        // in their natural order
        if (K == 0)
        {
            for (int i = 1; i <= N; i++)
            Console.Write(i + " ");
            Console.WriteLine();
            return;
        }

        // If N doesn't divide (2K) evenly, then
        // rearrangement is not possible
        if (N % (2 * K) != 0)
        {
            Console.Write("Not Possible\n");
            return;
        }

        // Copy first N numbers to an auxiliary
        // array
        int []arr=new int[N + 1];
        for (int i = 1; i <= N; i++)
            arr[i] = i;

        // Swap halves of each 2K sized slot
        for (int i = 1; i <= N; i += 2 * K)
            for (int j = 1; j <= K; j++)
            {
                int temp = arr[i + j - 1];
                arr[i + j - 1] = arr[K + i + j - 1];
                arr[K + i + j - 1] = temp;
            }

        // Print the rearranged array
        for (int i = 1; i <= N; i++)
            Console.Write(arr[i] + " ");
    }

    // Driver code
    public static void Main ()
    {
        int N = 12, K = 3;
        printRearrangeNnumberForKDistance(N, K);
    }
```

```
}

// This code is contributed by nitin mittal.
```

Output :

```
4 5 6 1 2 3 10 11 12 7 8 9
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/rearrange-first-n-numbers-make-k-distance/

Chapter 120

Rearrangement of a number which is also divisible by it

Rearrangement of a number which is also divisible by it - GeeksforGeeks

Given a number n, we need to rearrange all its digits such that the new arrangement is divisible by n. Also, the new number should not be equal to x. If no such rearrangement is possible print -1.

Examples:

```
Input : n = 1035
Output : 3105
The result 3105 is divisible by
given n and has same set of digits.

Input : n = 1782
Output : m = 7128
```

Simple Approach : Find all the permutation of given n and then check whether it is divisible by n or not also check that new permutation should not be equal to n.

Efficient Approach : Let's suppose that y is our result then y = m * n, also we know that y must be a rearrangement of digits of n so we can say now restrict m (the multiplier) as per given conditions.
1) y has the same number of digits as n has. So, m must be less than 10.
2) y must not be equal to n. So, m will be greater than 1.
So we get the multiplier m in the range [2,9]. So we will find all the possible y and then check that should y has the same digits as n or not.

```
 // CPP program for finding rearrangement of n
// that is divisible  by n
```

```
#include<bits/stdc++.h>
using namespace std;

// perform hashing for given n
void storeDigitCounts(int n, vector<int> &hash)
{
    // perform hashing
    while (n)
    {
        hash[n%10]++;
        n /= 10;
    }
}

// check whether any arrangement exists
int rearrange (int n)
{
    // Create a hash for given number n
    // The hash is of size 10 and stores
    // count of each digit in n.
    vector<int> hash_n(10, 0);
    storeDigitCounts(n, hash_n);

    // check for all possible multipliers
    for (int mult=2; mult<10; mult++)
    {
        int curr = n*mult;

        vector<int> hash_curr(10, 0);
        storeDigitCounts(curr, hash_curr);

        // check hash table for both.
        // Please refer below link for help
        // of equal()
        // https://www.geeksforgeeks.org/stdequal-in-cpp/
        if (equal(hash_n.begin(), hash_n.end(),
                             hash_curr.begin()))
            return curr;
    }
    return -1;
}

// driver program
int main()
{
    int n = 10035;
    cout << rearrange(n);
    return 0;
}
```

Output:

```
30105
```

Source

https://www.geeksforgeeks.org/rearrangement-number-also-divisible/

Chapter 121

Rencontres Number (Counting partial derangements)

Rencontres Number (Counting partial derangements) - GeeksforGeeks

Given two numbers, n >= 0 and 0 <= k <= n, count the number of derangements with k fixed points.

Examples:

```
Input : n = 3, k = 0
Output : 2
Since k = 0, no point needs to be on its
original position. So derangements
are {3, 1, 2} and {2, 3, 1}

Input : n = 3, k = 1
Output : 3
Since k = 1, one point needs to be on its
original position. So partial derangements
are {1, 3, 2}, {3, 2, 1} and {2, 1, 3}

Input : n = 7, k = 2
Output : 924
```

In combinatorial mathematics, the rencontres number< or D(n, k) represents count of partial derangements.

The recurrence relation to find Rencontres Number $D_{n,\ k}$:

$D_{(0,\ 0)} = 1$
$D_{(0,\ 1)} = 0$

$D_{(n+2,\ 0)} = (n+1) * (D_{(n+1,\ 0)} + D_{(n,\ 0)})$
$D_{(n,\ k)} = {}^{n}C_{k} * D_{(n-k,\ 0)})$

Given the two positive integer **n** and **k**. The task is find rencontres number D(n, k) for giver n and k.

Below is Recursive solution of this approach:

C++

```
 // Recursive CPP program to find n-th Rencontres
// Number
#include <bits/stdc++.h>
using namespace std;

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    // Base Cases
    if (k == 0 || k == n)
        return 1;

    // Recurrence relation
    return binomialCoeff(n - 1, k - 1) +
           binomialCoeff(n - 1, k);
}

// Return Recontres number D(n, m)
int RencontresNumber(int n, int m)
{
    // base condition
    if (n == 0 && m == 0)
        return 1;

    // base condition
    if (n == 1 && m == 0)
        return 0;

    // base condition
    if (m == 0)
        return (n - 1) * (RencontresNumber(n - 1, 0) +
                          RencontresNumber(n - 2, 0));

    return binomialCoeff(n, m) * RencontresNumber(n - m, 0);
}

// Driver Program
int main()
{
    int n = 7, m = 2;
    cout << RencontresNumber(n, m) << endl;
    return 0;
}
```

Java

```
 // Recursive Java program to find n-th Rencontres
// Number
import java.io.*;

class GFG {

    // Returns value of Binomial Coefficient
    // C(n, k)
    static int binomialCoeff(int n, int k)
    {

        // Base Cases
        if (k == 0 || k == n)
            return 1;

        // Recurrence relation
        return binomialCoeff(n - 1, k - 1) +
                         binomialCoeff(n - 1, k);
    }

    // Return Recontres number D(n, m)
    static int RencontresNumber(int n, int m)
    {

        // base condition
        if (n == 0 && m == 0)
            return 1;

        // base condition
        if (n == 1 && m == 0)
            return 0;

        // base condition
        if (m == 0)
            return (n - 1) * (RencontresNumber(n - 1, 0)
                          + RencontresNumber(n - 2, 0));

        return binomialCoeff(n, m) *
                              RencontresNumber(n - m, 0);
    }
    // Driver Program
    public static void main(String[] args)
    {
        int n = 7, m = 2;
        System.out.println(RencontresNumber(n, m));
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Recursive CPP program to find
# n-th Rencontres Number

# Returns value of Binomial Coefficient C(n, k)
def binomialCoeff(n, k):

    # Base Cases
    if (k == 0 or k == n):
        return 1

    # Recurrence relation
    return (binomialCoeff(n - 1, k - 1)
          + binomialCoeff(n - 1, k))

# Return Recontres number D(n, m)
def RencontresNumber(n, m):

    # base condition
    if (n == 0 and m == 0):
        return 1

    # base condition
    if (n == 1 and m == 0):
        return 0

    # base condition
    if (m == 0):
        return ((n - 1) * (RencontresNumber(n - 1, 0)
                         + RencontresNumber(n - 2, 0)))

    return (binomialCoeff(n, m) *
            RencontresNumber(n - m, 0))

# Driver Program
n = 7; m = 2
print(RencontresNumber(n, m))
```

```
# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // Recursive C# program to find n-th Rencontres
// Number
using System;

class GFG {

    // Returns value of Binomial Coefficient
    // C(n, k)
    static int binomialCoeff(int n, int k)
    {

        // Base Cases
        if (k == 0 || k == n)
            return 1;

        // Recurrence relation
        return binomialCoeff(n - 1, k - 1) +
                     binomialCoeff(n - 1, k);
    }

    // Return Recontres number D(n, m)
    static int RencontresNumber(int n, int m)
    {

        // base condition
        if (n == 0 && m == 0)
            return 1;

        // base condition
        if (n == 1 && m == 0)
            return 0;

        // base condition
        if (m == 0)
            return (n - 1) *
                (RencontresNumber(n - 1, 0)
                + RencontresNumber(n - 2, 0));

        return binomialCoeff(n, m) *
                 RencontresNumber(n - m, 0);
    }

    // Driver Program
    public static void Main()
    {
        int n = 7, m = 2;

        Console.Write(RencontresNumber(n, m));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

PHP

```
 <?php
// Recursive PHP program to
// find n-th Rencontres
// Number

// Returns value of Binomial
// Coefficient C(n, k)
function binomialCoeff($n, $k)
{

    // Base Cases
    if ($k == 0 || $k == $n)
        return 1;

    // Recurrence relation
    return binomialCoeff($n - 1,$k - 1) +
              binomialCoeff($n - 1, $k);
}

// Return Recontres number D(n, m)
function RencontresNumber($n, $m)
{

    // base condition
    if ($n == 0 && $m == 0)
        return 1;

    // base condition
    if ($n == 1 && $m == 0)
        return 0;

    // base condition
    if ($m == 0)
        return ($n - 1) * (RencontresNumber($n - 1, 0) +
                           RencontresNumber($n - 2, 0));
    return binomialCoeff($n, $m) *
           RencontresNumber($n - $m, 0);
}

    // Driver Code
    $n = 7;
    $m = 2;
    echo RencontresNumber($n, $m),"\n";

// This code is contributed by ajit.
?>
```

Output:

```
924
```

Below is the implementation using Dynamic Programming:

C++

```
 // DP based CPP program to find n-th Rencontres
// Number
#include <bits/stdc++.h>
using namespace std;
#define MAX 100

// Fills table C[n+1][k+1] such that C[i][j]
// represents table of binomial coefficient
// iCj
int binomialCoeff(int C[][MAX], int n, int k)
{
    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, k); j++) {

            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] +
                          C[i - 1][j];
        }
    }
}

// Return Recontres number D(n, m)
int RencontresNumber(int C[][MAX], int n, int m)
{
    int dp[n+1][m+1] = { 0 };

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (j <= i) {

                // base case
                if (i == 0 && j == 0)
                    dp[i][j] = 1;

                // base case
                else if (i == 1 && j == 0)
                    dp[i][j] = 0;

                else if (j == 0)
                    dp[i][j] = (i - 1) * (dp[i - 1][0] +
                                          dp[i - 2][0]);
                else
                    dp[i][j] = C[i][j] * dp[i - j][0];
            }
        }
    }

    return dp[n][m];
}

// Driver Program
int main()
{
    int n = 7, m = 2;

    int C[MAX][MAX];
    binomialCoeff(C, n, m);

    cout << RencontresNumber(C, n, m) << endl;
    return 0;
}
```

Java

```
 // DP based Java program to find n-th Rencontres
```

```
// Number

import java.io.*;

class GFG {

    static int MAX = 100;

    // Fills table C[n+1][k+1] such that C[i][j]
    // represents table of binomial coefficient
    // iCj
    static void binomialCoeff(int C[][], int n, int k)
    {

        // Calculate value of Binomial Coefficient
        // in bottom up manner
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= Math.min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i][j] = C[i - 1][j - 1] +
                                         C[i - 1][j];
            }
        }
    }

    // Return Recontres number D(n, m)
    static int RencontresNumber(int C[][], int n, int m)
    {
        int dp[][] = new int[n + 1][m + 1];

        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= m; j++) {
                if (j <= i) {

                    // base case
                    if (i == 0 && j == 0)
                        dp[i][j] = 1;

                    // base case
                    else if (i == 1 && j == 0)
```

```
                        dp[i][j] = 0;

                    else if (j == 0)
                        dp[i][j] = (i - 1) * (dp[i - 1][0]
                                          + dp[i - 2][0]);
                    else
                        dp[i][j] = C[i][j] * dp[i - j][0];
                }
            }
        }

        return dp[n][m];
    }

    // Driver Program
    public static void main(String[] args)
    {
        int n = 7, m = 2;

        int C[][] = new int[MAX][MAX];
        binomialCoeff(C, n, m);

        System.out.println(RencontresNumber(C, n, m));
    }
}

// This code is contributed by vt_m.
```

Python 3

```
 # DP based Python 3 program to find n-th
# Rencontres Number

MAX = 100

# Fills table C[n+1][k+1] such that C[i][j]
# represents table of binomial coefficient
# iCj
def binomialCoeff(C, n, k) :

    # Calculate value of Binomial Coefficient
    # in bottom up manner
    for i in range(0, n + 1) :
        for j in range(0, min(i, k) + 1) :

            # Base Cases
            if (j == 0 or j == i) :
                C[i][j] = 1
            # Calculate value using previously
            # stored values
            else :
                C[i][j] = (C[i - 1][j - 1]
                               + C[i - 1][j])


# Return Recontres number D(n, m)
def RencontresNumber(C, n, m) :
    w, h = m+1, n+1
    dp= [[0 for x in range(w)] for y in range(h)]


    for i in range(0, n+1) :
        for j in range(0, m+1) :
            if (j <= i) :

                # base case
                if (i == 0 and j == 0) :
                    dp[i][j] = 1

                # base case
                elif (i == 1 and j == 0) :
                    dp[i][j] = 0

                elif (j == 0) :
                    dp[i][j] = ((i - 1) *
                     (dp[i - 1][0] + dp[i - 2][0]))
                else :
                    dp[i][j] = C[i][j] * dp[i - j][0]

    return dp[n][m]


# Driver Program
n = 7
m = 2
C = [[0 for x in range(MAX)] for y in range(MAX)]

binomialCoeff(C, n, m)

print(RencontresNumber(C, n, m))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // DP based C# program
// to find n-th Rencontres
// Number
using System;

class GFG
{
    static int MAX = 100;

    // Fills table C[n+1][k+1]
    // such that C[i][j]
    // represents table of
    // binomial coefficient iCj
    static void binomialCoeff(int [,]C,
                              int n, int k)
    {

        // Calculate value of
        // Binomial Coefficient
        // in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0;
                     j <= Math.Min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using
                // previously stored values
                else
                    C[i, j] = C[i - 1, j - 1] +
                              C[i - 1, j];
            }
        }
    }

    // Return Recontres
    // number D(n, m)
    static int RencontresNumber(int [,]C,
                                int n, int m)
    {
        int [,]dp = new int[n + 1,
                            m + 1];

        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= m; j++)
            {
                if (j <= i)
                {

                    // base case
                    if (i == 0 && j == 0)
                        dp[i, j] = 1;

                    // base case
                    else if (i == 1 && j == 0)
                        dp[i, j] = 0;

                    else if (j == 0)
                        dp[i, j] = (i - 1) *
                                   (dp[i - 1, 0] +
                                    dp[i - 2, 0]);
                    else
                        dp[i, j] = C[i, j] *
                                 dp[i - j, 0];
                }
            }
        }

        return dp[n, m];
    }

    // Driver Code
    static public void Main ()
    {
        int n = 7, m = 2;
        int [,]C = new int[MAX, MAX];
        binomialCoeff(C, n, m);

        Console.WriteLine(RencontresNumber(C, n, m));
    }
}

// This code is contributed
// by akt_mit
```

Output:

```
924
```

Improved By : Nikita tiwari, Smitha Dinesh Semwal, jit_t

Source

https://www.geeksforgeeks.org/rencontres-number-counting-partial-derangements/

Chapter 122

Smallest Derangement of Sequence

Smallest Derangement of Sequence - GeeksforGeeks

Given the sequence [illegible] find the lexicographically smallest (earliest in dictionary order) derangement of [illegible].

A derangement of S is as any permutation of S such that no two elements in S and its permutation occur at same position.

Examples:

```
Input: 3
Output : 2 3 1
Explanation: The Sequence is 1 2 3.
Possible permutations are (1, 2, 3), (1, 3, 2),
          (2, 1, 3), (2, 3, 1), (3, 1, 2) (3, 2, 1).
Derangements are (2, 3, 1), (3, 1, 2).
Smallest Derangement: (2, 3, 1)

Input : 5
Output : 2 1 4 5 3.
Explanation: Out of all the permutations of
(1, 2, 3, 4, 5), (2, 1, 4, 5, 3) is the first derangement.
```

Method 1:
We can modify the method shown in this article: Largest Derangement
Using a min heap we can successively get the least element and place them in more significant positions, taking care that the property of derangement is maintained.
Complexity: O(N * log N)

Below is the C++ implementation.

```
 // CPP program to generate smallest derangement
// using priority queue.
#include <bits/stdc++.h>
using namespace std;

void generate_derangement(int N)
{
    // Generate Sequence and insert into a
    // priority queue.
    int S[N + 1];
    priority_queue<int, vector<int>, greater<int> > PQ;
    for (int i = 1; i <= N; i++) {
        S[i] = i;
        PQ.push(S[i]);
    }

    // Generate Least Derangement
    int D[N + 1];
    for (int i = 1; i <= N; i++) {
        int d = PQ.top();
        PQ.pop();
        if (d != S[i] || i == N) {
            D[i] = d;
        }
        else {
            D[i] = PQ.top();
            PQ.pop();
            PQ.push(d);
        }
    }

    if (D[N] == S[N])
       swap(S[N], D[N]);

    // Print Derangement
    for (int i = 1; i <= N; i++)
        printf("%d ", D[i]);
    printf("\n");
}

// Driver code
int main()
{
    generate_derangement(10);
    return 0;
}
```

Output:

```
2 1 4 3 6 5 8 7 10 9
```

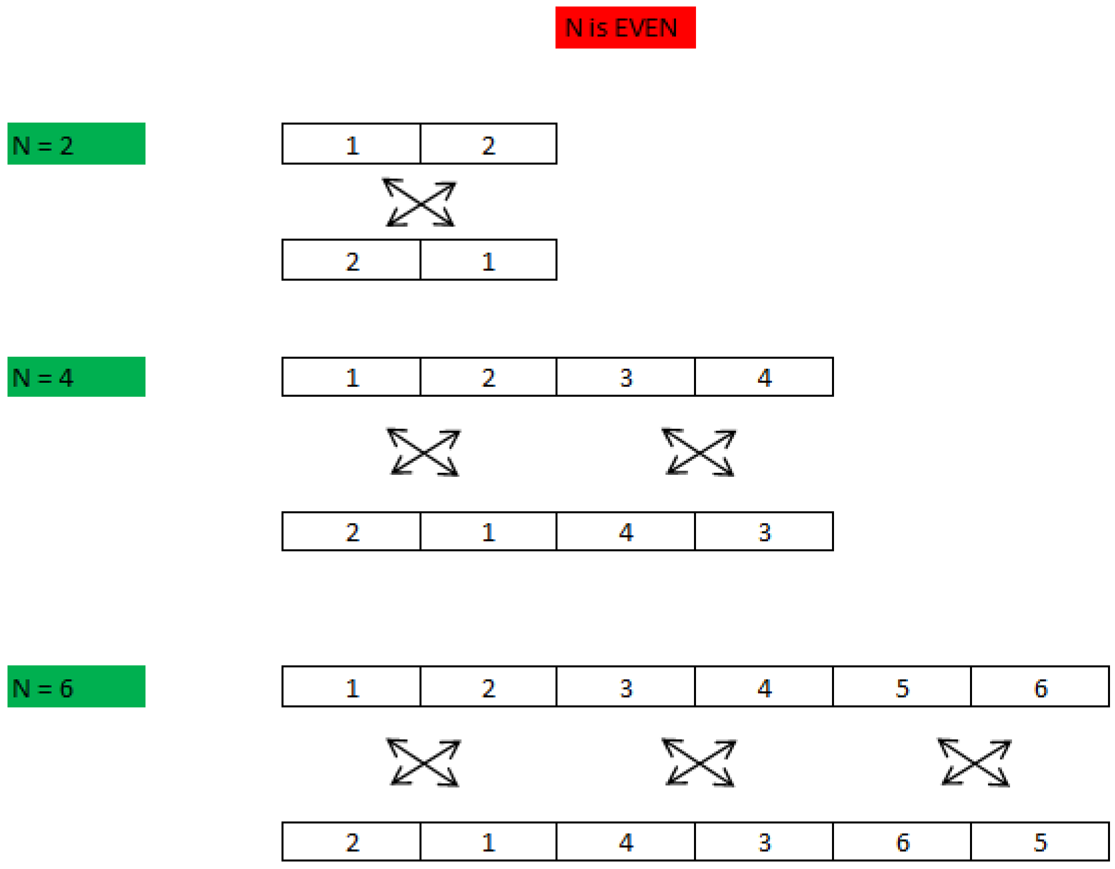
Method 2

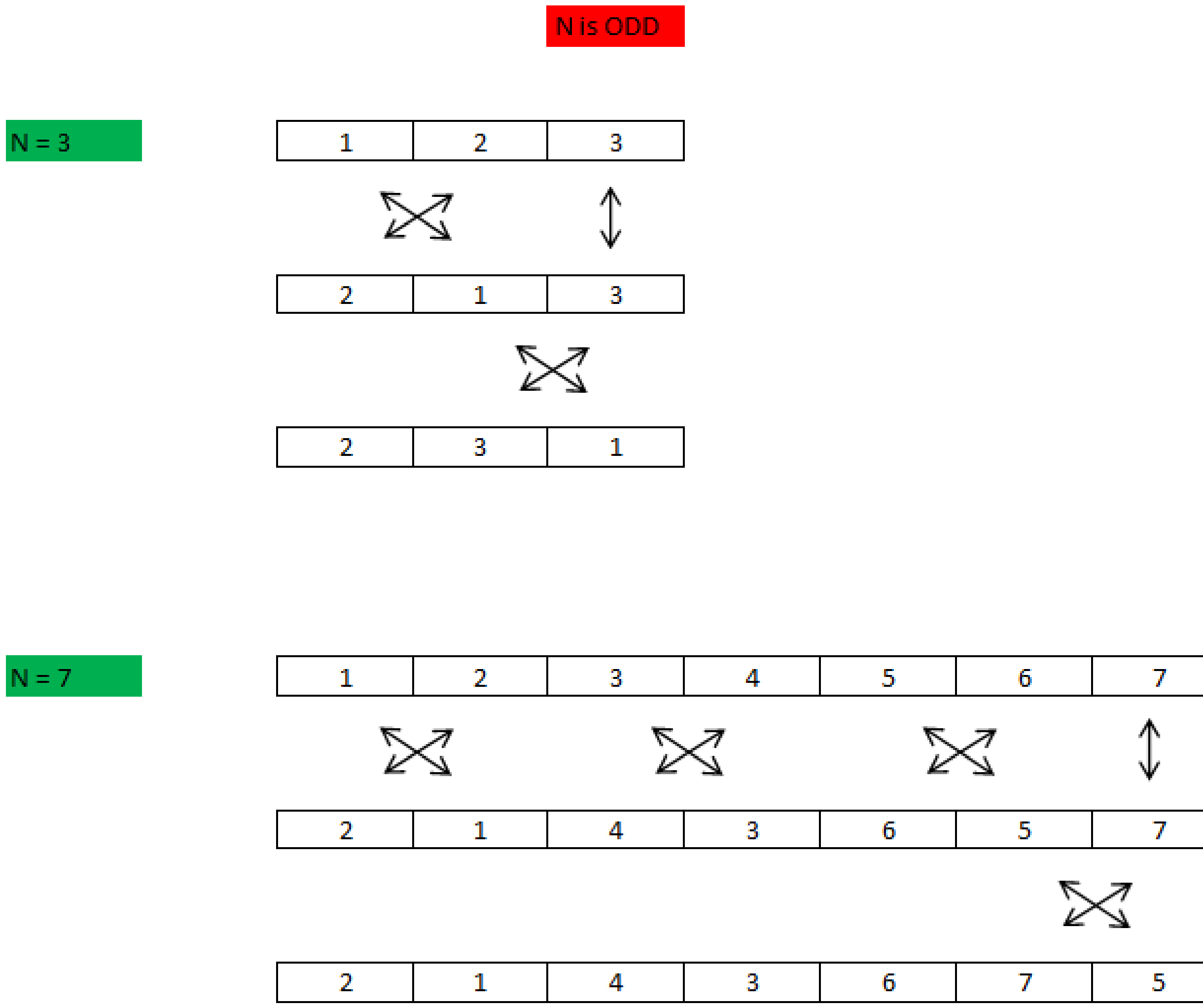
Since we are given a very specific sequence i.e [illegible] we can calculate the answer even more efficiently.

Divide the original sequence into pairs of two elements, and then swap the elements of each pair.

If N is odd then the last pair needs to be swapped again.

Pictorial Representation





Complexity: We perform at most N/2 + 1 swaps, so the complexity is O(N).

Why does this method work
This method is a very specific application of method 1 and is based on observation. Given the nature of sequence, at position i we already know the least element that can be put, which is either i+1 or i-1. Since we are already given the least permutation of S it is clear that the derangement must start from 2 and not 1 ie of the form i+1 (i = 1). The next element will be of the form i – 1 . The element after this will be i + 1 and then next i – 1. This pattern will continue until the end.

This operation is most easily understood as the swapping of adjacent elements of pairs of elements of S.

If we can determine the least element in constant time, then the complexity overhead from heap is eliminated. Hence from O(N * log N) the complexity reduces to O(N).

Below is the C++ implementation

```
 // Efficient C++ program to find smallest
// derangement.
#include <stdio.h>

void generate_derangement(int N)
{
    // Generate Sequence S
    int S[N + 1];
    for (int i = 1; i <= N; i++)
        S[i] = i;

    // Generate Derangement
    int D[N + 1];
    for (int i = 1; i <= N; i += 2) {
        if (i == N) {

            // Only if i is odd
            // Swap S[N-1] and S[N]
            D[N] = S[N - 1];
            D[N - 1] = S[N];
        }
        else {
            D[i] = i + 1;
            D[i + 1] = i;
        }
    }

    // Print Derangement
    for (int i = 1; i <= N; i++)
        printf("%d ", D[i]);
    printf("\n");
}

// Driver Program
int main()
{
    generate_derangement(10);
    return 0;
}
```

Output:

```
2 1 4 3 6 5 8 7 10 9
```

Note : The auxiliary space can be reduced to O(1) if we perform the swapping operations on S itself.

Source

https://www.geeksforgeeks.org/smallest-derangement-sequence/

Chapter 123

Stack Permutations (Check if an array is stack permutation of other)

Stack Permutations (Check if an array is stack permutation of other) - GeeksforGeeks

A **stack permutation** is a permutation of objects in the given input queue which is done by transferring elements from input queue to the output queue with the help of a stack and the built-in push and pop functions.

The well defined rules are:

1. Only dequeue from the input queue.
2. Use inbuilt push, pop functions in the single stack.
3. Stack and input queue must be empty at the end.
4. Only enqueue to the output queue.

There are a huge number of permutations possible using a stack for a single input queue. Given two arrays, both of unique elements. One represents the input queue and the other represents the output queue. Our task is to check if the given output is possible through stack permutation.

Examples:

```
Input : First array: 1, 2, 3
        Second array: 2, 1, 3
Output : Yes
Procedure:
push 1 from input to stack
push 2 from input to stack
```

```
pop 2 from stack to output
pop 1 from stack to output
push 3 from input to stack
pop 3 from stack to output


Input : First array: 1, 2, 3
        Second array: 3, 1, 2
Output : Not Possible
```

The idea to do this is we will try to convert the input queue to output queue using a stack, if we are able to do so then the queue is permutable otherwise not.
Below is the step by step algorithm to do this:

1. Continuously pop elements from the input queue and check if it is equal to the top of output queue or not, if it is not equal to the top of output queue then we will push the element to stack.
2. Once we find an element in input queue such the top of input queue is equal to top of output queue, we will pop a single element from both input and output queues, and compare the top of stack and top of output queue now. If top of both stack and output queue are equal then pop element from both stack and output queue. If not equal, go to step 1.
3. Repeat above two steps until the input queue becomes empty. At the end if both of the input queue and stack are empty then the input queue is permutable otherwise not.

Below is C++ implementation of above idea:

```
 // Given two arrays, check if one array is
// stack permutation of other.
#include<bits/stdc++.h>
using namespace std;

// function to check if input queue is
// permutable to output queue
bool checkStackPermutation(int ip[], int op[], int n)
{
    // Input queue
    queue<int> input;
    for (int i=0;i<n;i++)
        input.push(ip[i]);

    // output queue
    queue<int> output;
    for (int i=0;i<n;i++)
        output.push(op[i]);
    // stack to be used for permutation
    stack <int> tempStack;
    while (!input.empty())
    {
        int ele = input.front();
        input.pop();
        if (ele == output.front())
        {
            output.pop();
            while (!tempStack.empty())
            {
                if (tempStack.top() == output.front())
                {
                    tempStack.pop();
                    output.pop();
                }
                else
                    break;
            }
        }
        else
            tempStack.push(ele);
    }

    // If after processing, both input queue and
    // stack are empty then the input queue is
    // permutable otherwise not.
    return (input.empty()&&tempStack.empty());
}

// Driver program to test above function
int main()
{
    // Input Queue
    int input[] = {1, 2, 3};

    // Output Queue
    int output[] = {2, 1, 3};

    int n = 3;

    if (checkStackPermutation(input, output, n))
        cout << "Yes";
    else
        cout << "Not Possible";
    return 0;
}
```

Output:

```
Yes
```

Source

https://www.geeksforgeeks.org/stack-permutations-check-if-an-array-is-stack-permutation-of-other/

Chapter 124

Sum of Binomial coefficients

Sum of Binomial coefficients - GeeksforGeeks

Given a positive integer **n**, the task is to find the sum of binomail coefficient i.e

$^{n}C_{0} + {}^{n}C_{1} + {}^{n}C_{2} + \ldots\ldots + {}^{n}C_{n-1} + {}^{n}C_{n}$

Examples:

```
Input : n = 4
Output : 16
4C0 + 4C1 + 4C2 + 4C3 + 4C4
= 1 + 4 + 6 + 4 + 1
= 16

Input : n = 5
Output : 8
```

Method 1 (Brute Force):
The idea is to evaluate each binomial coefficient term i.e $^{n}C_{r}$, where 0 <= r <= n and calculate the sum of all the terms.

Below is the implementation of this approach:

C++

```
 // CPP Program to find the sum of Binomial
// Coefficient.
#include <bits/stdc++.h>
using namespace std;

// Returns value of Binomial Coefficient Sum
int binomialCoeffSum(int n)
```

```
{
    int C[n + 1][n + 1];

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, n); j++) {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    // Calculating the sum.
    int sum = 0;
    for (int i = 0; i <= n; i++)
        sum += C[n][i];

    return sum;
}

/* Driver program to test above function*/
int main()
{
    int n = 4;
    printf("%d", binomialCoeffSum(n));
    return 0;
}
```

Java

```
 // Java Program to find the sum
// of Binomial Coefficient.

class GFG {

    // Returns value of Binomial
    // Coefficient Sum
    static int binomialCoeffSum(int n)
    {
        int C[][] = new int[n + 1][n + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= Math.min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i][j] = C[i - 1][j - 1] +
                              C[i - 1][j];


            }
        }

        // Calculating the sum.
        int sum = 0;
        for (int i = 0; i <= n; i++)
            sum += C[n][i];

        return sum;
    }

    /* Driver program to test above function*/
    public static void main(String[] args)
    {
        int n = 4;
        System.out.println(binomialCoeffSum(n));
    }
}

// This code is contributed by prerna saini.
```

Python3

```
 # Python  Program to find the sum
# of Binomial Coefficient.

import math

# Returns value of Binomial
# Coefficient Sum
def binomialCoeffSum( n):
        C = [[0]*(n+2) for i in range(0,n+2)]

        # Calculate value of Binomial
        # Coefficient in bottom up manner
        for i in range(0,n+1):
            for j in range(0, min(i, n)+1):

                # Base Cases
                if (j == 0 or j == i):
                    C[i][j] = 1

                # Calculate value using previously
                # stored values
                else:
                    C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

        # Calculating the sum.
        sum = 0
        for i in range(0,n+1):
            sum += C[n][i]

        return sum


# Driver program to test above function
n = 4
print(binomialCoeffSum(n))

# This code is contributed by Gitanjali.
```

C#

```
 // C# program to find the sum
// of Binomial Coefficient.
using System;

class GFG {

    // Returns value of Binomial
    // Coefficient Sum
    static int binomialCoeffSum(int n)
    {
        int[, ] C = new int[n + 1, n + 1];

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= Math.Min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i, j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i, j] = C[i - 1, j - 1] + C[i - 1, j];
            }
        }

        // Calculating the sum.
        int sum = 0;
        for (int i = 0; i <= n; i++)
            sum += C[n, i];

        return sum;
    }

    /* Driver program to test above function*/
    public static void Main()
    {
        int n = 4;
        Console.WriteLine(binomialCoeffSum(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the
// sum of Binomial Coefficient.
// Returns value of Binomial
// Coefficient Sum

function binomialCoeffSum($n)
{
    $C[$n + 1][$n + 1] = array(0);

    // Calculate value of
    // Binomial Coefficient
    // in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0;
             $j <= min($i, $n); $j++)
        {
            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value
            // using previously
            // stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1] +
                             $C[$i - 1][$j];
        }
    }

    // Calculating the sum.
    $sum = 0;
    for ($i = 0; $i <= $n; $i++)
        $sum += $C[$n][$i];

    return $sum;
}

// Driver Code
$n = 4;
echo binomialCoeffSum($n);

// This code is contributed by ajit
?>
```

Output:

```
16
```

Method 2 (Using Formula):

This can be proved in 2 ways.
First Proof: Using Principle of induction.

> For basic step, n = 0
> LHS = ${}^0C_0$ = (0!)/(0! * 0!) = 1/1 = 1.
> RHS= 2^0 = 1.
> LHS = RHS

For induction step:
Let k be an integer such that k > 0 and for all r, 0 <= r <= k, where r belong to integers,
the formula stand true.
Therefore,
$^{k}C_{0} + {}^{k}C_{1} + {}^{k}C_{2} + \ldots\ldots. + {}^{k}C_{k-1} + {}^{k}C_{k} = 2^{k}$

Now, we have to prove for n = k + 1,
$^{k+1}C_{0} + {}^{k+1}C_{1} + {}^{k+1}C_{2} + \ldots\ldots. + {}^{k+1}C_{k} + {}^{k+1}C_{k+1} = 2^{k+1}$

LHS = $^{k+1}C_{0} + {}^{k+1}C_{1} + {}^{k+1}C_{2} + \ldots\ldots. + {}^{k+1}C_{k} + {}^{k+1}C_{k+1}$
(Using $^{n}C_{0} = 0$ and $^{n+1}C_{r} = {}^{n}C_{r} + {}^{n}C_{r-1}$)
$= 1 + {}^{k}C_{0} + {}^{k}C_{1} + {}^{k}C_{1} + {}^{k}C_{2} + \ldots\ldots + {}^{k}C_{k-1} + {}^{k}C_{k} + 1$
$= {}^{k}C_{0} + {}^{k}C_{0} + {}^{k}C_{1} + {}^{k}C_{1} + \ldots\ldots + {}^{k}C_{k-1} + {}^{k}C_{k-1} + {}^{k}C_{k} + {}^{k}C_{k}$
$= 2 \text{ X } {}^{n}C_{r}$
$= 2 \text{ X } 2^{k}$
$= 2^{k+1}$
= RHS

Second Proof: Using Binomial theorem expansion

Binomial expansion state,
$(x + y)^{n} = {}^{n}C_{0}\, x^{n}\, y^{0} + {}^{n}C_{1}\, x^{n-1}\, y^{1} + {}^{n}C_{2}\, x^{n-2}\, y^{2} + \ldots\ldots\ldots + {}^{n}C_{n-1}\, x^{1}\, y^{n-1} + {}^{n}C_{n}\, x^{0}\, y^{n}$

Put x = 1, y = 1
$(1 + 1)^{n} = {}^{n}C_{0}\, 1^{n}\, 1^{0} + {}^{n}C_{1}\, x^{n-1}\, 1^{1} + {}^{n}C_{2}\, 1^{n-2}\, 1^{2} + \ldots\ldots\ldots + {}^{n}C_{n-1}\, 1^{1}\, 1^{n-1} + {}^{n}C_{n}\, 1^{0}\, 1^{n}$

$2^{n} = {}^{n}C_{0} + {}^{n}C_{1} + {}^{n}C_{2} + \ldots\ldots. + {}^{n}C_{n-1} + {}^{n}C_{n}$

Below is implementation of this approach:

C++

```
 // CPP Program to find sum of Binomial
// Coefficient.
#include <bits/stdc++.h>
using namespace std;

// Returns value of Binomial Coefficient Sum
// which is 2 raised to power n.
int binomialCoeffSum(int n)
{
    return (1 << n);
}

/* Drier program to test above function*/
int main()
```

```
{
    int n = 4;
    printf("%d", binomialCoeffSum(n));
    return 0;
}
```

Java

```
 // Java Program to find sum
// of Binomial Coefficient.
import java.io.*;

class GFG
{
    // Returns value of Binomial
    // Coefficient Sum which is
    // 2 raised to power n.
    static int binomialCoeffSum(int n)
    {
        return (1 << n);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 4;
        System.out.println(binomialCoeffSum(n));
    }
}

// This code is contributed
// by akt_mit.
```

Python3

```
 # Python  Program to find the sum
# of Binomial Coefficient.

import math
# Returns value of Binomial
# Coefficient Sum
def binomialCoeffSum( n):

    return (1 << n);

# Driver program to test
# above function
```

```
n = 4
print(binomialCoeffSum(n))

# This code is contributed
# by Gitanjali.
```

C#

```
 // C# Program to find sum of
// Binomial Coefficient.
using System;

class GFG {

    // Returns value of Binomial Coefficient Sum
    // which is 2 raised to power n.
    static int binomialCoeffSum(int n)
    {
        return (1 << n);
    }

    /* Drier program to test above function*/
    static public void Main()
    {
        int n = 4;
        Console.WriteLine(binomialCoeffSum(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find sum
// of Binomial Coefficient.

// Returns value of Binomial
// Coefficient Sum which is
// 2 raised to power n.
function binomialCoeffSum($n)
{
    return (1 << $n);
}

// Driver Code
$n = 4;
```

```
echo binomialCoeffSum($n);

// This code is contributed
// by akt_mit
?>
```

Output:

```
16
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/sum-binomial-coefficients/

Chapter 125

Sum of all numbers that can be formed with permutations of n digits

Sum of all numbers that can be formed with permutations of n digits - GeeksforGeeks

Given n distinct digits (from 0 to 9), find sum of all n digit numbers that can be formed using these digits. It is assumed that numbers formed with leading 0 are allowed.

Example:

```
Input: 1 2 3
Output: 1332
Explanation
Numbers Formed: 123 , 132 , 312 , 213, 231 , 321
123 + 132 + 312 + 213 + 231 + 321 = 1332
```

Total numbers that can be formed using n digits is total number of permutations of n digits, i.e factorial(n). Now, since the number formed is a n-digit number, each digit will appear factorial(n)/n times at each position from least significant digit to most significant digit. Therefore, sum of digits at a position = (sum of all the digits) * (factorial(n)/n).

```
Considering the example digits as 1 2 3

factorial(3)/3 = 2

Sum of digits at least significant digit = (1 + 2 + 3) * 2 = 12

Similarly sum of digits at tens, hundreds place is 12.
```

```
(This sum will contribute as 12 * 100)

Similarly sum of digits at tens, thousands place is 12.
(This sum will contribute as 12 * 1000)

Required sum of all numbers = 12 + (10 * 12) + (100 * 12) = 1332
```

C++

```
 // C++ program to find sun of numbers formed
// by all permutations of given set of digits
#include<stdio.h>

// function to calculate factorial of a number
int factorial(int n)
{
    int f = 1;
    if (n==0||n==1)
        return 1;
    for (int i=2; i<=n; i++)
        f = f*i;
    return f;
}

// Function to calculate sum of all numbers
int getSum(int arr[],int n)
{
    // calculate factorial
    int fact = factorial(n);

    // sum of all the given digits at different
    // positions is same and is going to be stored
    // in digitsum.
    int digitsum = 0;
    for (int i=0; i<n; i++)
        digitsum += arr[i];
    digitsum *= (fact/n);

    // Compute result (sum of all the numbers)
    int res = 0;
    for (int i=1, k=1; i<=n; i++)
    {
        res  += (k*digitsum);
        k = k*10;
    }

    return res;
}
```

```
// Driver program to test above function
int main()
{
    // n distinct digits
    int arr[] = {1, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Print sum of all the numbers formed
    printf("%d", getSum(arr, n));

    return 0;
}
```

Java

```
 // Java program to find sum
// of numbers formed by all
// permutations of given set
// of digits
import java.io.*;

class GFG
{

// function to calculate
// factorial of a number
static int factorial(int n)
{
    int f = 1;
    if (n == 0|| n == 1)
        return 1;
    for (int i = 2; i <= n; i++)
        f = f * i;
    return f;
}

// Function to calculate
// sum of all numbers
static int getSum(int arr[], int n)
{
    // calculate factorial
    int fact = factorial(n);

    // sum of all the given
    // digits at different
    // positions is same and
    // is going to be stored
    // in digitsum.
    int digitsum = 0;
    for (int i = 0; i < n; i++)
        digitsum += arr[i];
    digitsum *= (fact / n);

    // Compute result (sum
    // of all the numbers)
    int res = 0;
    for (int i = 1, k = 1;
             i <= n; i++)
    {
        res += (k * digitsum);
        k = k * 10;
    }

    return res;
}

// Driver Code
public static void main (String[] args)
{

    // n distinct digits
    int arr[] = {1, 2, 3};
    int n = arr.length;

    // Print sum of all
    // the numbers formed
    System.out.println(getSum(arr, n));
}
}

// This code is contributed
// by ajit
```

C#

```
 // C# program to find sum
// of numbers formed by all
// permutations of given set
// of digits
using System;

class GFG
{

// function to calculate
// factorial of a number
static int factorial(int n)
{
    int f = 1;
    if (n == 0|| n == 1)
        return 1;
    for (int i = 2; i <= n; i++)
        f = f * i;
    return f;
}

// Function to calculate
// sum of all numbers
static int getSum(int []arr,
                  int n)
{
    // calculate factorial
    int fact = factorial(n);

    // sum of all the given
    // digits at different
    // positions is same and
    // is going to be stored
    // in digitsum.
    int digitsum = 0;
    for (int i = 0; i < n; i++)
        digitsum += arr[i];
    digitsum *= (fact / n);

    // Compute result (sum
    // of all the numbers)
    int res = 0;
    for (int i = 1, k = 1;
            i <= n; i++)
    {
        res += (k * digitsum);
        k = k * 10;
    }

    return res;
}

// Driver Code
static public void Main ()
{

    // n distinct digits
    int []arr = {1, 2, 3};
    int n = arr.Length;

    // Print sum of all
    // the numbers formed
    Console.WriteLine(getSum(arr, n));
}
}

// This code is contributed
// by akt_mit
```

PHP

```
 <?php
// PHP program to find sum
// of numbers formed by all
// permutations of given set
// of digits function to
// calculate factorial of a number
function factorial($n)
{
    $f = 1;
    if ($n == 0||$n == 1)
        return 1;
    for ($i = 2; $i <= $n; $i++)
        $f = $f * $i;
    return $f;
}

// Function to calculate
// sum of all numbers
function getSum($arr,$n)
{
    // calculate factorial
    $fact = factorial($n);

    // sum of all the given
    // digits at different
    // positions is same and
    // is going to be stored
    // in digitsum.
    $digitsum = 0;
    for ($i = 0; $i < $n; $i++)
        $digitsum += $arr[$i];
    $digitsum *= ($fact / $n);

    // Compute result (sum
    // of all the numbers)
    $res = 0;
    for ($i = 1, $k = 1; $i <= $n; $i++)
    {
        $res += ($k * $digitsum);
        $k = $k * 10;
    }

    return $res;
}

// Driver Code

// n distinct digits
$arr = array(1, 2, 3);
$n = sizeof($arr);

// Print sum of all
// the numbers formed
echo getSum($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
1332
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/sum-numbers-can-formed-permutations-n-digits/

Chapter 126

Sum of products of all combination taken (1 to n) at a time

Sum of products of all combination taken (1 to n) at a time - GeeksforGeeks

Given N, we have to find the sum of products of all combination taken 1 to N at a time. In simple words, we have to find the sum of products of all combination taken 1 at time, then 2 at a time, then 3 at a time till N at a time.

If you think closely about the problem, a large value of N could result in producing a large number of combinations.

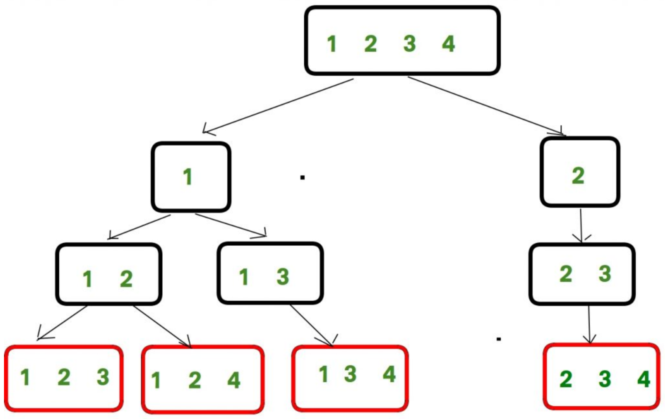
Examples:

```
Input :  N = 3
Output : f(1) = 6
         f(2) = 11
         f(3) = 6

Explanation: f(x) is sum of products of all
             combination taken x at a time
             1 + 2 + 3 = 6
             f(2) = (1*2) + (1*3) + (2*3) = 11
             f(3) = (1*2*3)

Input :  N = 4
Output : f(1) = 10
         f(2) = 35
         f(3) = 50
         f(4) = 24
```

```
Explanation: f(1) = 1 + 2 + 3 + 4 = 10
             f(2) = (1*2) + (1*3) + (1*4) +
                    (2*3) + (2*4) + (3*4)
                  = 35
             f(3) = (1*2*3) + (1*2*4) +(1*3*4) +
                    (2*3*4)
                 = 50
             f(4) = (1*2*3*4) = 24
```

A **Brute force** approach would be to produce all the combinations and then find their products and sum.

Recursion would do the trick to produce the combinations taken x at a time.
Example : N = 4 taken 3 at a time



C++

```
 // Program to find SOP of all combination taken
// (1 to N) at a time using brute force
#include <iostream>
using namespace std;

// to store sum of every combination
int sum = 0;

void Combination(int a[], int combi[], int n,
                int r, int depth, int index) {

  // if we have reached sufficient depth
  if (index == r) {

    // find the product of combination
    int product = 1;
    for (int i = 0; i < r; i++)
      product = product * combi[i];

    // add the product into sum
    sum += product;
    return;
  }

  // recursion to produce different combination
  for (int i = depth; i < n; i++) {
    combi[index] = a[i];
    Combination(a, combi, n, r, i + 1, index + 1);
  }
}

// function to print sum of products of
// all combination taken 1-N at a time
void allCombination(int a[], int n) {
  for (int i = 1; i <= n; i++) {

    // creating temporary array for storing
    // combination
    int *combi = new int[i];

    // call combination with r = i
    // for combination taken i at a time
    Combination(a, combi, n, i, 0, 0);

    // displaying sum
    cout << "f(" << i << ") --> " << sum << "\n";
    sum = 0;

    // free from heap area
    free(combi);
  }
}

// Driver's code
int main() {
  int n = 5;
  int *a = new int[n];

  // storing numbers from 1-N in array
  for (int i = 0; i < n; i++)
    a[i] = i + 1;

  // calling allCombination
  allCombination(a, n);

  return 0;
}
```

Java

```
 // Program to find SOP of
// all combination taken
// (1 to N) at a time using
// brute force
import java.io.*;

class GFG
{
    // to store sum of
    // every combination
    static int sum = 0;

    static void Combination(int []a, int []combi,
                            int n, int r,
                            int depth, int index)
    {

    // if we have reached
    // sufficient depth
    if (index == r)
    {

        // find the product
        // of combination
        int product = 1;
        for (int i = 0; i < r; i++)
        product = product * combi[i];

        // add the product into sum
        sum += product;
        return;
    }

    // recursion to produce
    // different combination
    for (int i = depth; i < n; i++)
    {
        combi[index] = a[i];
        Combination(a, combi, n, r,
                    i + 1, index + 1);
    }
    }

    // function to print sum of
    // products of all combination
    // taken 1-N at a time
    static void allCombination(int []a,
                               int n)
    {
        for (int i = 1; i <= n; i++)
        {

            // creating temporary array
            // for storing combination
            int []combi = new int[i];

            // call combination with
            // r = i for combination
            // taken i at a time
            Combination(a, combi, n,
                        i, 0, 0);

            // displaying sum
            System.out.print("f(" + i + ") --> " +
                                       sum + "\n");
            sum = 0;
        }
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 5;
        int []a = new int[n];

        // storing numbers
        // from 1-N in array
        for (int i = 0; i < n; i++)
            a[i] = i + 1;

        // calling allCombination
        allCombination(a, n);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // Program to find SOP of
// all combination taken
// (1 to N) at a time using
// brute force
using System;

class GFG
{
    // to store sum of
    // every combination
    static int sum = 0;

    static void Combination(int []a, int []combi,
                            int n, int r,
                            int depth, int index)
    {

    // if we have reached
    // sufficient depth
    if (index == r)
    {

        // find the product
        // of combination
        int product = 1;
        for (int i = 0; i < r; i++)
        product = product * combi[i];

        // add the product into sum
        sum += product;
        return;
    }

    // recursion to produce
    // different combination
    for (int i = depth; i < n; i++)
    {
        combi[index] = a[i];
        Combination(a, combi, n, r,
                    i + 1, index + 1);
    }
    }

    // function to print sum of
    // products of all combination
    // taken 1-N at a time
    static void allCombination(int []a,
                               int n)
    {
    for (int i = 1; i <= n; i++)
    {

        // creating temporary array
        // for storing combination
        int []combi = new int[i];

        // call combination with
        // r = i for combination
        // taken i at a time
        Combination(a, combi, n,
                    i, 0, 0);

        // displaying sum
        Console.Write("f(" + i + ") --> " +
                               sum + "\n");
        sum = 0;
    }
    }

    // Driver code
    static void Main()
    {
        int n = 5;
        int []a = new int[n];

        // storing numbers
        // from 1-N in array
        for (int i = 0; i < n; i++)
            a[i] = i + 1;

        // calling allCombination
        allCombination(a, n);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

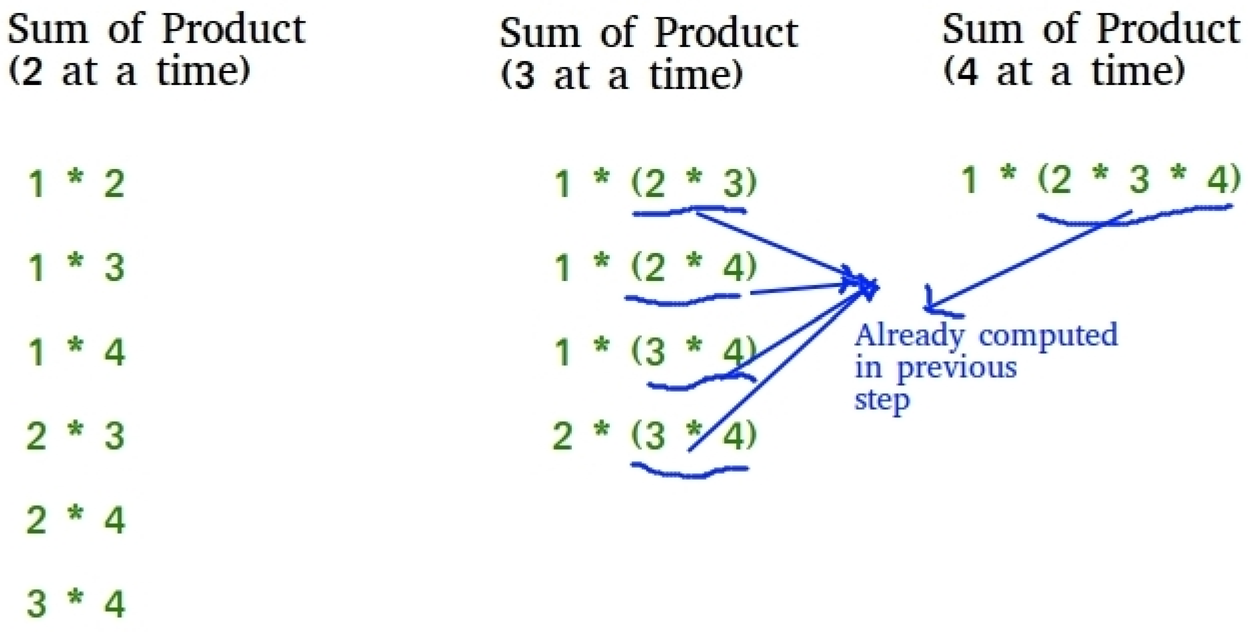
Output:

```
f(1) --> 15
f(2) --> 85
f(3) --> 225
f(4) --> 274
```

```
f(5) --> 120
```

The **Time complexity** of above code is exponential when the value of N is large.

An **Efficient Method** is to use the concept of dynamic programming. We don't have to find sum of products every time. We can make use of previous results.
Let's take an example: N = 4


Sum of Product (2 at a time)
1 * 2
1 * 3
1 * 4
2 * 3
2 * 4
3 * 4
Sum of Product (3 at a time)
1 * (2 * 3)
1 * (2 * 4)
1 * (3 * 4)
2 * (3 * 4)
Sum of Product (4 at a time)
1 * (2 * 3 * 4)
Already computed in previous step

C++

```
 // CPP Program to find sum of all combination takne
// (1 to N) at a time using dynamic programming
#include <iostream>
using namespace std;

// find the postfix sum array
void postfix(int a[], int n) {
  for (int i = n - 1; i > 0; i--)
    a[i - 1] = a[i - 1] + a[i];
}

// modify the array such that we don't have to
// compute the products which are obtained before
void modify(int a[], int n) {
  for (int i = 1; i < n; i++)
    a[i - 1] = i * a[i];
}

// finding sum of all combination taken 1 to N at a time
void allCombination(int a[], int n) {

  int sum = 0;

  // sum taken 1 at time is simply sum of 1 - N
  for (int i = 1; i <= n; i++)
    sum += i;
  cout << "f(1) --> " << sum << "\n";

  // for sum of products for all combination
  for (int i = 1; i < n; i++) {

    // finding postfix array
    postfix(a, n - i + 1);

    // sum of products taken i+1 at a time
    sum = 0;
    for (int j = 1; j <= n - i; j++) {
      sum += (j * a[j]);
    }
    cout << "f(" << i + 1 << ") --> " << sum << "\n";

    // modify the array for overlapping problem
    modify(a, n);
  }
}

// Driver's Code
int main() {
  int n = 5;
  int *a = new int[n];

  // storing numbers from 1 to N
  for (int i = 0; i < n; i++)
    a[i] = i + 1;

  // calling allCombination
  allCombination(a, n);

  return 0;
}
```

Output:

```
f(1) --> 15
f(2) --> 85
f(3) --> 225
f(4) --> 274
f(5) --> 120
```

The **Time Complexity** of above method is O(n^2) which far more better than the brute force method.
You can also find the execution time of both the method for large value of N and can see the difference for yourself.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/sum-products-combination-taken-1-n-time/

Chapter 127

Sum of squares of binomial coefficients

Sum of squares of binomial coefficients - GeeksforGeeks

Given a positive integer **n**. The task is to find the sum of square of Binomial Coefficient i.e
$\mathbf{{}^{n}C_{0}^{2} + {}^{n}C_{1}^{2} + {}^{n}C_{2}^{2} + {}^{n}C_{3}^{2} + \ldots\ldots\ldots + {}^{n}C_{n-2}^{2} + {}^{n}C_{n-1}^{2} + {}^{n}C_{n}^{2}}$

Examples:

```
Input : n = 4
Output : 70

Input : n = 5
Output : 252
```

Method 1: (Brute Force)
The idea is to generate all the terms of binomial coefficient and find the sum of square of each binomial coefficient.

Below is the implementation of this approach:

C++

```
 // CPP Program to find the sum of square of
// binomial coefficient.
#include<bits/stdc++.h>
using namespace std;

// Return the sum of square of binomial coefficient
int sumofsquare(int n)
{
    int C[n+1][n+1];
    int i, j;

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= min(i, n); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }

    // Finding the sum of square of binomial
    // coefficient.
    int sum = 0;
    for (int i = 0; i <= n; i++)
        sum += (C[n][i] * C[n][i]);

    return sum;
}

// Driven Program
int main()
{
    int n = 4;
    cout << sumofsquare(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to find the sum of
// square of binomial coefficient.
import static java.lang.Math.*;

class GFG{

    // Return the sum of square of
    // binomial coefficient
    static int sumofsquare(int n)
    {
        int[][] C = new int [n+1][n+1] ;
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++)
        {
            for (j = 0; j <= min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using
                //previously stored values
                else
                    C[i][j] = C[i-1][j-1]
                             + C[i-1][j];
            }
        }

        // Finding the sum of square of
        // binomial coefficient.
        int sum = 0;
        for (i = 0; i <= n; i++)
            sum += (C[n][i] * C[n][i]);

        return sum;
    }

    // Driver function
    public static void main(String[] args)
    {
        int n = 4;

        System.out.println(sumofsquare(n));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
 # Python Program to find
# the sum of square of
# binomial coefficient.
# Return the sum of
# square of binomial
# coefficient
def sumofsquare(n) :

    C = [[0 for i in range(n + 1)]
            for j in range(n + 1)]

    # Calculate value of
    # Binomial Coefficient
    # in bottom up manner
    for i in range(0, n + 1) :

        for j in range(0, min(i, n) + 1) :

            # Base Cases
            if (j == 0 or j == i) :
                C[i][j] = 1

            # Calculate value
            # using previously
            # stored values
            else :
                C[i][j] = (C[i - 1][j - 1] +
                           C[i - 1][j])


    # Finding the sum of
    # square of binomial
    # coefficient.
    sum = 0
    for i in range(0, n + 1) :
        sum = sum + (C[n][i] *
                     C[n][i])

    return sum


# Driver Code
n = 4
print (sumofsquare(n), end="\n")

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# Program to find the sum of
// square of binomial coefficient.
using System;

class GFG {

    // Return the sum of square of
    // binomial coefficient
    static int sumofsquare(int n)
    {
        int[,] C = new int [n+1,n+1] ;
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++)
        {
            for (j = 0; j <= Math.Min(i, n); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using
                //previously stored values
                else
                    C[i,j] = C[i-1,j-1]
                            + C[i-1,j];
            }
        }

        // Finding the sum of square of
        // binomial coefficient.
        int sum = 0;
        for (i = 0; i <= n; i++)
            sum += (C[n,i] * C[n,i]);

        return sum;
    }

    // Driver function
    public static void Main()
    {
        int n = 4;

        Console.WriteLine(sumofsquare(n));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the sum of
// square of binomial coefficient.

// Return the sum of square of binomial
// coefficient
function sumofsquare($n)
{
    $i; $j;

    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++)
    {
        for ($j = 0; $j <= min($i, $n); $j++)
        {

            // Base Cases
            if ($j == 0 || $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using previously
            // stored values
            else
                $C[$i][$j] = $C[$i-1][$j-1]
                                + $C[$i-1][$j];
        }
    }

    // Finding the sum of square of binomial
    // coefficient.
    $sum = 0;
    for ($i = 0; $i <= $n; $i++)
        $sum += ($C[$n][$i] * $C[$n][$i]);

    return $sum;
}

// Driven Program
    $n = 4;
    echo sumofsquare($n), "\n";

// This code is contributed by ajit
```

```
?>
```

Output:

```
70
```

Method 2: (Using Formula)

[illegible]

= [illegible]

= [illegible]

Proof,

```
We know,
(1 + x)n = nC0 + nC1 x + nC2 x2 + ......... + nCn-1 xn-1 + nCn-1 xn
Also,
(x + 1)n = nC0 xn + nC1 xn-1 + nC2 xn-2 + ......... + nCn-1 x + nCn

Multiplying above two equations,
(1 + x)2n = [nC0 + nC1 x + nC2 x2 + ......... + nCn-1 xn-1 + nCn-1 xn] X
            [nC0 xn + nC1 xn-1 + nC2 xn-2 + ......... + nCn-1 x + nCn]

Equating coefficients of xn on both sides, we get
2nCn = nC02 + nC12 + nC22 + nC32 + ......... + nCn-22 + nCn-12 + nCn2

Hence, sum of the squares of coefficients = 2nCn = (2n)!/(n!)2.
```

Also, $(2n)!/(n!)^2$ = (2n * (2n – 1) * (2n – 2) * ..... * (n+1))/(n * (n – 1) * (n – 2) *..... * 1).

Below is the implementation of this approach:

C++

```
 // CPP Program to find the sum of square of
// binomial coefficient.
#include<bits/stdc++.h>
using namespace std;

// function to return product of number
// from start to end.
int factorial(int start, int end)
{
    int res = 1;
    for (int i = start; i <= end; i++)
        res *= i;

    return res;
}

// Return the sum of square of binomial
// coefficient
int sumofsquare(int n)
{
    return factorial(n+1, 2*n)/factorial(1, n);
}

// Driven Program
int main()
{
    int n = 4;
    cout << sumofsquare(n) << endl;
    return 0;
}
```

Java

```
 // Java Program to find the sum of square of
// binomial coefficient.
class GFG{

    // function to return product of number
    // from start to end.
    static int factorial(int start, int end)
    {
        int res = 1;
        for (int i = start; i <= end; i++)
            res *= i;

        return res;
    }

    // Return the sum of square of binomial
    // coefficient
    static int sumofsquare(int n)
    {
        return factorial(n+1, 2*n)/factorial(1, n);
    }

    // Driven Program
    public static void main(String[] args)
    {
        int n = 4;
        System.out.println(sumofsquare(n));
    }
}

// This code is contributed by
// Smitha DInesh Semwal
```

Python

```
 # Python 3 Program to find the sum of
# square of binomial coefficient.

# function to return product of number
# from start to end.
def factorial(start, end):

    res = 1

    for i in range(start, end + 1):
        res *= i

    return res

# Return the sum of square of binomial
# coefficient
def sumofsquare(n):

    return int(factorial(n + 1, 2 * n)
                     /factorial(1, n))

# Driven Program

n = 4
print(sumofsquare(n))


# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# Program to find the sum of square of
// binomial coefficient.
using System;

class GFG {
    // function to return product of number
    // from start to end.
    static int factorial(int start, int end)
    {
        int res = 1;

        for (int i = start; i <= end; i++)
            res *= i;

        return res;
    }

    // Return the sum of square of binomial
    // coefficient
    static int sumofsquare(int n)
    {
        return factorial(n+1, 2*n)/factorial(1, n);
    }

    // Driven Program
    public static void Main()
    {
        int n = 4;

        Console.WriteLine(sumofsquare(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the sum
// of square of binomial coefficient.

// function to return
// product of number
// from start to end.
function factorial($start, $end)
{
    $res = 1;
    for ($i = $start;
         $i <= $end; $i++)
        $res *= $i;

    return $res;
}

// Return the sum of
// square of binomial
// coefficient
function sumofsquare($n)
{
    return factorial($n + 1,
                     2 * $n) /
           factorial(1, $n);
}

// Driver Code
$n = 4;
echo sumofsquare($n), "\n";

// This code is contributed by ajit
?>
```

Output:

```
70
```

Improved By : jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/sum-squares-binomial-coefficients/

Chapter 128

Telephone Number

Telephone Number - GeeksforGeeks

In mathematics, the **telephone numbers involution numbers** are a sequence of integers are a sequence of integers that count the number of connection patterns in a telephone system with **n** subscribers, where connections are made between pairs of subscribers. These numbers also describe the number of matchings of a complete graph of n vertices, the number of permutations on n elements that are involutions, the sum of absolute value of coefficients of the Hermite polynomials, the number of standard Young tableaux with n cells, and the sum of the degrees of the irreducible representations of the symmetric group.

The telephone numbers are also used to count the number of ways to place **n** rooks on an **n x n** chessboard in such a way that no two rooks attack each other and in such a way the configuration of the rooks is symmetric under a diagonal reflection of the board.

The telephone number can be evaluated by the following recurrence relation:

$$T(n) = \begin{cases} 1 & n=0 \text{ or } n=1 \\ T(n-1) + (n-1) * T(n-2) & \text{otherwise} \end{cases}$$

Given a positive integer **n**. The task is to find the nth telephone number.
Examples :

```
Input : n = 4
Output : 10

Input : n = 6
Output : 76
```

Below is naive implementation of finding the nth telephone number based on above recursive formula.

C++

```
 // CPP Program to find the nth telephone number.
#include <bits/stdc++.h>
using namespace std;

// return the nth telephone number
int telephonenumber(int n)
{
    // base step
    if (n == 0 || n == 1)
        return 1;

    // recursive step
    return telephonenumber(n - 1) +
          (n - 1) * telephonenumber(n - 2);
}

// Driven Program
int main()
{
    int n = 6;
    cout << telephonenumber(n) << endl;
    return 0;
}
```

Java

```
 // JAVA Code to find the nth
// telephone number.
import java.util.*;

class GFG {

    // return the nth telephone number
    static int telephonenumber(int n)
    {
        // base step
        if (n == 0 || n == 1)
            return 1;

        // recursive step
        return telephonenumber(n - 1) +
              (n - 1) * telephonenumber(n - 2);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 6;
        System.out.println(telephonenumber(n));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 code to find the
# nth telephone number.

# return the nth telephone number
def telephonenumber (n):

    # base step
    if n == 0 or n == 1:
        return 1

    # recursive step
    return (telephonenumber(n - 1) + (n - 1)
            * telephonenumber(n - 2))

# Driven Program
n = 6
print(telephonenumber(n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code to find the nth
// telephone number.
using System;

class GFG {

    // return the nth telephone number
    static int telephonenumber(int n)
    {
        // base step
        if (n == 0 || n == 1)
            return 1;

        // recursive step
        return telephonenumber(n - 1) +
            (n - 1) * telephonenumber(n - 2);
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 6;

        Console.Write(telephonenumber(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// the nth telephone number

// return the nth
// telephone number
function telephonenumber( $n)
{
    // base step
    if ($n == 0 or $n == 1)
        return 1;

    // recursive step
    return telephonenumber($n - 1) +
        ($n - 1) * telephonenumber($n - 2);
}

// Driven Code
$n = 6;
echo telephonenumber($n) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
76
```

Below is efficient implementation of finding the nth telephone number using Dynamic Programming:

C++

```
 // CPP Program to find the nth telephone number.
#include <bits/stdc++.h>
using namespace std;

// return the nth telephone number
int telephonenumber(int n)
{
    int dp[n + 1];
    memset(dp, 0, sizeof(dp));

    // Base case
    dp[0] = dp[1] = 1;

    // finding ith telephone number, where 2 <= i <= n.
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + (i - 1) * dp[i - 2];

    return dp[n];
}

// Driver Program
int main()
{
    int n = 6;
    cout << telephonenumber(n) << endl;
    return 0;
}
```

Java

```
 // JAVA Code to find nth Telephone Number
import java.util.*;

class GFG {

    // return the nth telephone number
    static int telephonenumber(int n)
    {
        int dp[] = new int[n + 1];

        // Base case
        dp[0] = dp[1] = 1;

        // finding ith telephone number,
        // where 2 <= i <= n.
        for (int i = 2; i <= n; i++)
            dp[i] = dp[i - 1] + (i - 1) * dp[i - 2];

        return dp[n];
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
         int n = 6;
         System.out.println(telephonenumber(n));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 code to find the
# nth telephone number.

# return the nth telephone number
def telephonenumber (n):
    dp = [0] * (n + 1)

    # Base case
    dp[0] = dp[1] = 1

    # finding ith telephone number,
    # where 2 <= i <= n.
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + (i - 1) * dp[i - 2]

    return dp[n]

# Driver Code
n = 6
print(telephonenumber(n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Code to find nth Telephone Number
using System;

class GFG {
    // return the nth telephone number
    static int telephonenumber(int n)
    {
        int[] dp = new int[n + 1];

        // Base case
        dp[0] = dp[1] = 1;

        // finding ith telephone number,
        // where 2 <= i <= n.
        for (int i = 2; i <= n; i++)
            dp[i] = dp[i - 1] + (i - 1) * dp[i - 2];

        return dp[n];
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int n = 6;

        Console.Write(telephonenumber(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// the nth telephone number.

// return the nth telephone number
function telephonenumber($n)
{
    $dp = array();

    // Base case
    $dp[0] = $dp[1] = 1;

    // finding ith telephone number,
    // where 2 <= i <= n.
    for ( $i = 2; $i <= $n; $i++)
        $dp[$i] = $dp[$i - 1] +
                     ($i - 1) *
                    $dp[$i - 2];
    return $dp[$n];
}

// Driver Code
$n = 6;
echo telephonenumber($n);

// This code is contributed by anuj_67.
?>
```

Output :

```
76
```

Improved By : vt_m

Source

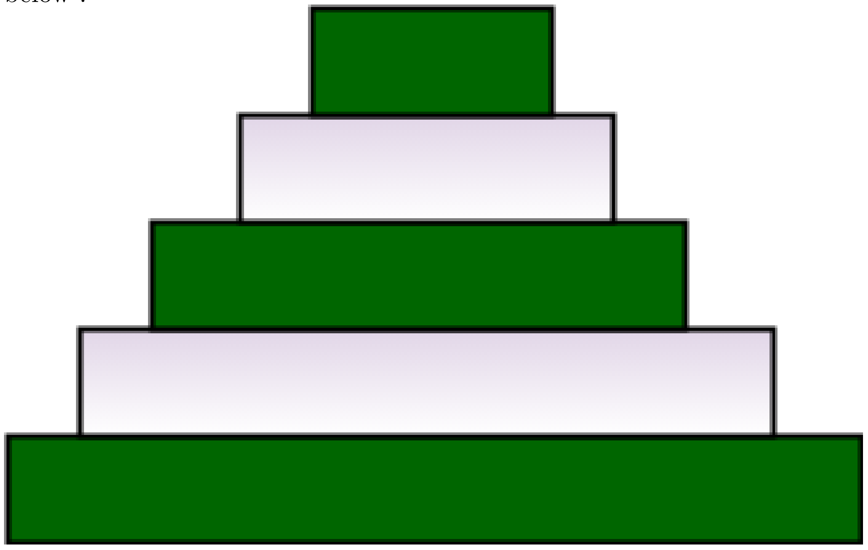
https://www.geeksforgeeks.org/telephone-number/

Chapter 129

Tile Stacking Problem

Tile Stacking Problem - GeeksforGeeks

A stable tower of height **n** is a tower consisting of exactly n tiles of unit height stacked vertically in such a way, that no bigger tile is placed on a smaller tile. An example is shown below :



We have infinite number of tiles of sizes 1, 2, …, m. The task is calculate the number of different stable tower of height n that can be built from these tiles, with a restriction that you can use at most **k** tiles of each size in the tower.

Note: Two tower of height n are different if and only if there exists a height h (1 <= h <= n), such that the towers have tiles of different sizes at height h.

Examples:

```
Input : n = 3, m = 3, k = 1.
```

```
Output : 1
Possible sequences: { 1, 2, 3}.
Hence answer is 1.

Input : n = 3, m = 3, k = 1.
Output : 7
{1, 1, 2}, {1, 1, 3}, {1, 2, 2},
{1, 2, 3}, {1, 3, 3}, {2, 2, 3},
{2, 3, 3}.
```

We basically need to count number of decreasing sequences of length n using numbers from 1 to m where every number can be used at most k times. We can recursively compute count for n using count for n-1.

The idea is to use Dynamic Programming. Declare a 2D array dp[][], where each state dp[i][j] denotes the number of decreasing sequences of length i using numbers from j to m. We need to take care of the fact that a number can be used a most k times. This can be done by considering 1 to k occurrences of a number. Hence our recurrence relation becomes:

$$DP[i][j] = \sum_{x=0}^{k} dp[i - x][j - 1]$$

Also, we can use the fact that for a fixed j we are using the consecutive values of previous k values of i. Hence, we can maintain a prefix sum array for each state. Now we have got rid of the k factor for each state.

Below is the C++ implemantation of this approach:

```
 // CPP program to find number of ways to make stable
// tower of given height.
#include <bits/stdc++.h>
using namespace std;
#define N 100

int possibleWays(int n, int m, int k)
{
    int dp[N][N];
    int presum[N][N];
    memset(dp, 0, sizeof dp);
    memset(presum, 0, sizeof presum);

    // Initialing 0th row to 0.
    for (int i = 1; i < n + 1; i++) {
        dp[0][i] = 0;
        presum[0][i] = 1;
    }

    // Initialing 0th column to 0.
    for (int i = 0; i < m + 1; i++)
        presum[i][0] = dp[i][0] = 1;
    // For each row from 1 to m
    for (int i = 1; i < m + 1; i++) {

        // For each column from 1 to n.
        for (int j = 1; j < n + 1; j++) {

            // Initialing dp[i][j] to presum of (i - 1, j).
            dp[i][j] = presum[i - 1][j];
            if (j > k) {
                dp[i][j] -= presum[i - 1][j - k - 1];
            }
        }

        // Calculating presum for each i, 1 <= i <= n.
        for (int j = 1; j < n + 1; j++)
            presum[i][j] = dp[i][j] + presum[i][j - 1];
    }

    return dp[m][n];
}

// Driver Program
int main()
{
    int n = 3, m = 3, k = 2;
    cout << possibleWays(n, m, k) << endl;
    return 0;
}
```

Output:

```
7
```

Source

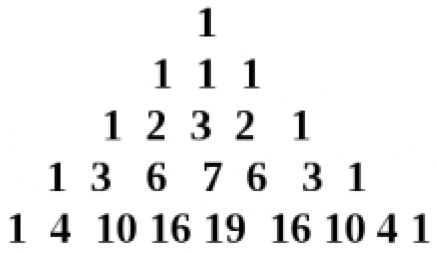
https://www.geeksforgeeks.org/tile-stacking-problem/

Chapter 130

Trinomial Triangle

Trinomial Triangle - GeeksforGeeks

The **trinomial triangle** is a variation of Pascal's triangle. The difference between the two is that an entry in the trinomial triangle is the sum of the three (rather than the two in Pasacal's triangle) entries above it :



```
            1
          1 1 1
        1 2 3 2 1
      1 3 6 7 6 3 1
1 4 10 16 19 16 10 4 1
```

The **k**-th entry of the **n**-th row is denoted by :

$\binom{n}{k}_2$

Rows are counted starting from 0. The entries of the **n**-th row are indexed starting with **-n** from the left, and the middle entry has index 0. The symmetry of the entries of a row about the middle entry is expressed by the relationship

$\binom{n}{k}_2 = \binom{n}{-k}_2$

Properties :

- The **n**-th row corresponds to the coefficients in the polynomial expansion of the expansion of the trinomial $\mathbf{(1 + x + x^2)}$ raised to the **n**-th power.

  $(1 + x + x^2)^n = \sum_{j=0}^{2n} \binom{n}{j-n}_2 x^j = \sum_{k=-n}^{n} \binom{n}{k}_2 x^{n+k}$

  or Symmentrically

[illegible]

hence the alternative name **trinomial coefficients** because of their relationship to the multinomial coefficients :

[illegible]

- The diagonals have intersecting properties, such as their relationship to the triangular numbers.

- The sum of the elements of **n**-th row is $\mathbf{3^n}$.

Recursion formula
The trinomial coefficients can be generated using the following recursion formula :

[illegible]

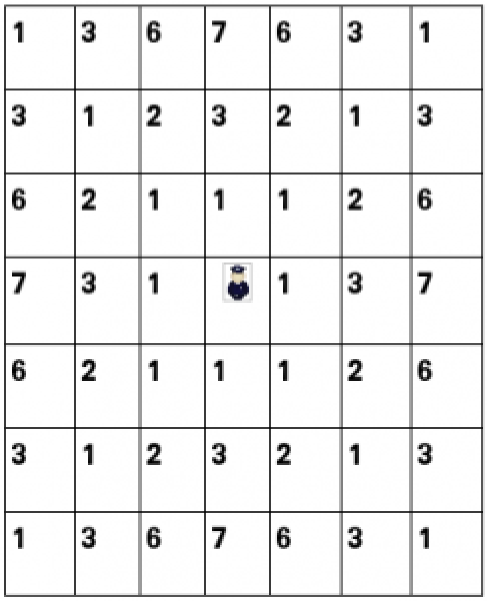
[illegible]

where,

$\binom{n}{k} = 0$ **, for k n**

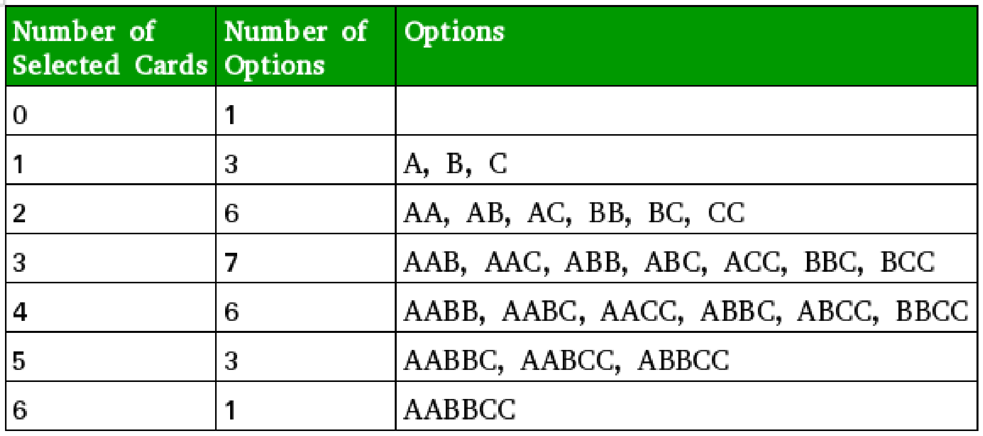
Applications :

- The triangle corresponds to the number of possible paths that can be taken by the king in a game of chess. The entry in a cell represents the number of different paths (using minimum number of moves) the king can take to reach the cell.



1	3	6	7	6	3	1
3	1	2	3	2	1	3
6	2	1	1	1	2	6
7	3	1	♚	1	3	7
6	2	1	1	1	2	6
3	1	2	3	2	1	3
1	3	6	7	6	3	1

- The coefficient of $\mathbf{x^k}$ in the polynomial $\mathbf{(1 + x + x^2)^n}$ specifies the number of different ways of randomly drawing **k** cards from two sets of **n** identical playing cards. For example, in such a card game with two sets of the three cards A, B, C , the choices look like this :



Number of Selected Cards	Number of Options	Options
0	1	
1	3	A, B, C
2	6	AA, AB, AC, BB, BC, CC
3	7	AAB, AAC, ABB, ABC, ACC, BBC, BCC
4	6	AABB, AABC, AACC, ABBC, ABCC, BBCC
5	3	AABBC, AABCC, ABBCC
6	1	AABBCC

Given a positive number **n**. The task is to print Trinomial Triangle of height n.
Examples:

```
Input : n = 4
Output :
1
1 1 1
1 2 3 2 1
1 3 6 7 6 3 1

Input : n = 5
Output :
1
1 1 1
1 2 3 2 1
1 3 6 7 6 3 1
1 4 10 16 19 16 10 4 1
```

Below is the implementation of printing trinomial triangle og height n :

C++

```
 // CPP Program to print trinomial triangle.
#include<bits/stdc++.h>
using namespace std;

// Function to find the trinomial triangle value.
int TrinomialValue(int n, int k)
```

```
{
    // base case
    if (n == 0 && k == 0)
        return 1;

    // base case
    if(k < -n || k > n)
        return 0;

    // recursive step.
    return TrinomialValue (n - 1, k - 1)
           + TrinomialValue (n - 1, k)
           + TrinomialValue (n - 1, k + 1);
}

// Function to print Trinomial Triangle of height n.
void printTrinomial(int n)
{
    // printing n rows.
    for (int i = 0; i < n; i++)
    {
        // printing first half of triangle
        for (int j = -i; j <= 0; j++)
            cout << TrinomialValue(i, j) << " ";

        // printing second half of triangle.
        for (int j = 1; j <= i; j++)
            cout << TrinomialValue(i, j) << " ";

        cout << endl;
    }
}

// Driven Program
int main()
{
    int n = 4;

    printTrinomial(n);
    return 0;
}
```

Java

```
 // Java Program to print trinomial triangle.
import java.util.*;
import java.lang.*;
```

```
public class GfG {

    // Function to find the trinomial
    // triangle value.
    public static int TrinomialValue(int n,
                                     int k)
    {
        // base case
        if (n == 0 && k == 0)
            return 1;

        // base case
        if (k < -n || k > n)
            return 0;

        // recursive step.
        return TrinomialValue(n - 1, k - 1)
            + TrinomialValue(n - 1, k)
            + TrinomialValue(n - 1, k + 1);
    }

    // Function to print Trinomial
    // Triangle of height n.
    public static void printTrinomial(int n)
    {
        // printing n rows.
        for (int i = 0; i < n; i++)
        {
            // printing first half of triangle
            for (int j = -i; j <= 0; j++)
                System.out.print(TrinomialValue(i, j)
                                    + " ");

            // printing second half of triangle.
            for (int j = 1; j <= i; j++)
                System.out.print(TrinomialValue(i, j)
                                  + " ");

            System.out.println();
        }
    }

    // driver function
    public static void main(String argc[])
    {
        int n = 4;

        printTrinomial(n);
    }

}

/* This code is contributed by Sagar Shukla */
```

Python3

```
 # Python3 code to print trinomial triangle.

# Function to find the trinomial triangle value.
def TrinomialValue(n, k):
    # base case
    if n == 0 and k == 0:
        return 1

    # base cas
    if k < -n or k > n:
        return 0

    # recursive step.
    return (TrinomialValue (n - 1, k - 1)+
                TrinomialValue (n - 1, k)+
                        TrinomialValue (n - 1, k + 1))

# Function to print Trinomial Triangle of height n.
def printTrinomial( n ):

    # printing n rows.
    for i in range(n):

        # printing first half of triangle
        for j in range(-i, 1):
            print(TrinomialValue(i, j),end=" ")

        # printing second half of triangle.
        for j in range(1, i+1):
            print( TrinomialValue(i, j),end=" ")

        print("\n",end='')

# Driven Code
n = 4
printTrinomial(n)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Program to print trinomial triangle.
using System;

public class GfG {

    // Function to find the trinomial
    // triangle value.
    public static int TrinomialValue(int n,
                                int k)
    {
        // base case
        if (n == 0 && k == 0)
            return 1;

        // base case
        if (k < -n || k > n)
            return 0;

        // recursive step.
        return TrinomialValue(n - 1, k - 1)
            + TrinomialValue(n - 1, k)
            + TrinomialValue(n - 1, k + 1);
    }

    // Function to print Trinomial
    // Triangle of height n.
    public static void printTrinomial(int n)
    {
        // printing n rows.
        for (int i = 0; i < n; i++)
        {
            // printing first half of triangle
            for (int j = -i; j <= 0; j++)
                Console.Write(TrinomialValue(i, j)
                                + " ");

            // printing second half of triangle.
            for (int j = 1; j <= i; j++)
                Console.Write(TrinomialValue(i, j)
                            + " ");

            Console.WriteLine();
        }
    }

    // Driver function
    public static void Main()
    {
        int n = 4;

        printTrinomial(n);
    }

}

/* This code is contributed by Vt_m */
```

PHP

```
 <?php
// PHP Program to print
// trinomial triangle.

// Function to find the
// trinomial triangle value.
function TrinomialValue($n, $k)
{
    // base case
    if ($n == 0 && $k == 0)
        return 1;

    // base case
    if($k < -$n || $k > $n)
        return 0;

    // recursive step.
    return TrinomialValue ($n - 1, $k - 1) +
           TrinomialValue ($n - 1, $k) +
           TrinomialValue ($n - 1, $k + 1);
}

// Function to print Trinomial
// Triangle of height n.
function printTrinomial($n)
{
    // printing n rows.
    for ($i = 0; $i < $n; $i++)
    {
        // printing first
        // half of triangle
        for ($j = -$i; $j <= 0; $j++)
            echo TrinomialValue($i, $j), " ";

        // printing second
        // half of triangle.
        for ($j = 1; $j <= $i; $j++)
```

```
            echo TrinomialValue($i, $j) , " ";

        echo "\n";
    }
}

// Driver Code
$n = 4;

printTrinomial($n);

// This code is contributed
// by ajit
?>
```

Output:

```
1
1 1 1
1 2 3 2 1
1 3 6 7 6 3 1
```

Below is the implementation of printing Trinomial Triangle using Dynamic Programming and property of trinomial triangle i.e $\binom{n}{k} = \binom{n}{-k}_2$

C

```
 // CPP Program to print trinomial triangle.
#include<bits/stdc++.h>
#define MAX 10
using namespace std;

// Function to find the trinomial triangle value.
int TrinomialValue(int dp[MAX][MAX], int n, int k)
{
    // Using property of trinomial triangle.
    if (k < 0)
        k = -k;

    // If value already calculated, return that.
    if (dp[n][k] != 0)
        return dp[n][k];

    // base case
    if (n == 0 && k == 0)
        return 1;
    // base case
    if(k < -n || k > n)
        return 0;

    // recursive step and storing the value.
    return (dp[n][k] = TrinomialValue(dp, n - 1, k - 1)
           + TrinomialValue(dp, n - 1, k)
           + TrinomialValue(dp, n - 1, k + 1));
}

// Function to print Trinomial Triangle of height n.
void printTrinomial(int n)
{
    int dp[MAX][MAX] = { 0 };

    // printing n rows.
    for (int i = 0; i < n; i++)
    {
        // printing first half of triangle
        for (int j = -i; j <= 0; j++)
            cout << TrinomialValue(dp, i, j) << " ";

        // printing second half of triangle.
        for (int j = 1; j <= i; j++)
            cout << TrinomialValue(dp, i, j) << " ";

        cout << endl;
    }
}

// Driven Program
int main()
{
    int n = 4;

    printTrinomial(n);
    return 0;
}
```

Java

```
 // Java Program to print trinomial triangle.
import java.util.*;
import java.lang.*;

public class GfG {
```

```
private static final int MAX = 10;

// Function to find the trinomial triangle value.
public static int TrinomialValue(int dp[][], int n, int k)
{
    // Using property of trinomial triangle.
    if (k < 0)
        k = -k;

    // If value already calculated, return that.
    if (dp[n][k] != 0)
        return dp[n][k];

    // base case
    if (n == 0 && k == 0)
        return 1;

    // base case
    if (k < -n || k > n)
        return 0;

    // recursive step and storing the value.
    return (dp[n][k] = TrinomialValue(dp, n - 1, k - 1)
                     + TrinomialValue(dp, n - 1, k)
                     + TrinomialValue(dp, n - 1, k + 1));
}

// Function to print Trinomial Triangle of height n.
public static void printTrinomial(int n)
{
    int[][] dp = new int[MAX][MAX];

    // printing n rows.
    for (int i = 0; i < n; i++) {
        // printing first half of triangle
        for (int j = -i; j <= 0; j++)
            System.out.print(TrinomialValue(dp, i, j) + " ");

        // printing second half of triangle.
        for (int j = 1; j <= i; j++)
            System.out.print(TrinomialValue(dp, i, j) + " ");

        System.out.println();
    }
}

// driver function
public static void main(String argc[])
```

```
    {
        int n = 4;
        printTrinomial(n);
    }

}
/* This code is contributed by Sagar Shukla */
```

Python3

```
 # Python3 code to print trinomial triangle.

# Function to find the trinomial triangle value.
def TrinomialValue(dp , n , k):

    # Using property of trinomial triangle.
    if k < 0:
        k = -k

    # If value already calculated, return that.
    if dp[n][k] != 0:
        return dp[n][k]

    # base case
    if n == 0 and k == 0:
        return 1

    # base case
    if k < -n or k > n:
        return 0

    # recursive step and storing the value.
    return  (TrinomialValue(dp, n - 1, k - 1) +
                TrinomialValue(dp, n - 1, k)+
                    TrinomialValue(dp, n - 1, k + 1))

# Function to print Trinomial Triangle of height n.
def printTrinomial(n):
    dp = [[0]*10]*10

    # printing n rows.
    for i in range(n):

        # printing first half of triangle
        for j in range(-i,1):
            print(TrinomialValue(dp, i, j),end=" ")

        # printing second half of triangle.
        for j in range(1,i+1):
            print(TrinomialValue(dp, i, j),end=" ")
        print("\n",end='')

# Driven Program
n = 4
printTrinomial(n)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Program to print
// trinomial triangle.
using System;

class GFG
{

    private static int MAX = 10;

    // Function to find the
    // trinomial triangle value.
    public static int TrinomialValue(int [,]dp,
                                     int n, int k)
    {
        // Using property of
        // trinomial triangle.
        if (k < 0)
            k = -k;

        // If value already
        // calculated, return that.
        if (dp[n, k] != 0)
            return dp[n, k];

        // base case
        if (n == 0 && k == 0)
            return 1;

        // base case
        if (k < -n || k > n)
            return 0;

        // recursive step and storing the value.
        return (dp[n, k] = TrinomialValue(dp, n - 1,
                                          k - 1) +
                           TrinomialValue(dp, n - 1,
                                                  k) +
                               TrinomialValue(dp, n - 1,
                                              k + 1));
    }

    // Function to print Trinomial
    // Triangle of height n.
    public static void printTrinomial(int n)
    {
        int[,] dp = new int[MAX, MAX];

        // printing n rows.
        for (int i = 0; i < n; i++)
        {
            // printing first
            // half of triangle
            for (int j = -i; j <= 0; j++)
            Console.Write(TrinomialValue(dp, i,
                                         j) + " ");

            // printing second half
            // of triangle.
            for (int j = 1; j <= i; j++)
                Console.Write(TrinomialValue(dp, i,
                                             j) + " ");

            Console.WriteLine();
        }
    }

    // Driver Code
    static public void Main ()
    {
        int n = 4;
        printTrinomial(n);
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP Program to print
// trinomial triangle.

$MAX = 10;
```

```
// Function to find the
// trinomial triangle value.
function TrinomialValue($dp, $n, $k)
{
    // Using property of
    // trinomial triangle.
    if ($k < 0)
        $k = -$k;

    // If value already
    // calculated, return that.
    if ($dp[$n][$k] != 0)
        return $dp[$n][$k];

    // base case
    if ($n == 0 && $k == 0)
        return 1;

    // base case
    if($k < -$n || $k > $n)
        return 0;

    // recursive step and
    // storing the value.
    return ($dp[$n][$k] = TrinomialValue($dp, $n - 1, $k - 1) +
                          TrinomialValue($dp, $n - 1, $k) +
                          TrinomialValue($dp, $n - 1, $k + 1));
}

// Function to print Trinomial
// Triangle of height n.
function printTrinomial($n)
{
    global $MAX;
    $dp;
    for ($i = 0; $i < $MAX; $i++)
    for ($j = 0; $j < $MAX; $j++)
        $dp[$i][$j] = 0;

    // printing n rows.
    for ($i = 0; $i < $n; $i++)
    {
        // printing first
        // half of triangle
        for ($j = -$i; $j <= 0; $j++)
            echo TrinomialValue($dp, $i, $j)." ";

        // printing second
        // half of triangle.
        for ($j = 1; $j <= $i; $j++)
            echo TrinomialValue($dp, $i, $j)." ";

        echo "\n";
    }
}

// Driven Code
$n = 4;
printTrinomial($n);

// This code is contributed by mits
?>
```

Output:

```
1
1 1 1
1 2 3 2 1
1 3 6 7 6 3 1
```

Improved By : jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/trinomial-triangle/

Chapter 131

Ways of filling matrix such that product of all rows and all columns are equal to unity

Ways of filling matrix such that product of all rows and all columns are equal to unity - GeeksforGeeks

We are given three values n, m and k where n is number of rows in matrix, m is number of columns in the matrix and k is the number that can have only two values -1 and 1. Our aim is to find the number of ways of filling the matrix of $n \times m$ such that the product of all the elements in each row and each column is equal to k. Since the number of ways can be large we will output $ways \mod 1000000007$
Examples:

```
Input : n = 2, m = 4, k = -1
Output : 8
Following configurations satisfy the conditions:-
```

```
Input  : n = 2, m = 1, k = -1
Output : The number of filling the matrix
         are 0
```

From the above conditions, it is clear that the only elements that can be entered in the matrix are 1 and -1. Now we can easily deduce some of the corner cases

1. If k = -1, then the sum of number of rows and columns cannot be odd because -1 will

be present odd number of times in each row and column therefore if the sum is odd then answer is 0.
2. If n = 1 or m = 1 then there is only one way of filling the matrix therefore answer is 1.
3. If none of the above cases are applicable then we fill the first $n - 1$ rows and the first $m - 1$ columns with 1 and -1. Then the remaining numbers can be uniquely identified since the product of each row an each column is already known therefore the answer is [illegible].

C++

```
 // CPP program to find number of ways to fill
// a matrix under given constraints
#include <bits/stdc++.h>
using namespace std;

#define mod 100000007

/* Returns a raised power t under modulo mod */
long long modPower(long long a, long long t)
{
    long long now = a, ret = 1;

    // Counting number of ways of filling the matrix
    while (t) {
        if (t & 1)
            ret = now * (ret % mod);
        now = now * (now % mod);
        t >>= 1;
    }
    return ret;
}

// Function calculating the answer
long countWays(int n, int m, int k)
{
    // if sum of numbers of rows and columns is odd
    // i.e (n + m) % 2 == 1 and k = -1 then there
    // are 0 ways of filiing the matrix.
    if (k == -1 && (n + m) % 2 == 1)
        return 0;

    // If there is one row or one column then there
    // is only one way of filling the matrix
    if (n == 1 || m == 1)
        return 1;
    // If the above cases are not followed then we
    // find ways to fill the n - 1 rows and m - 1
    // columns which is 2 ^ ((m-1)*(n-1)).
    return (modPower(modPower((long long)2, n - 1),
                                    m - 1) % mod);
}

// Driver function for the program
int main()
{
    int n = 2, m = 7, k = 1;
    cout << countWays(n, m, k);
    return 0;
}
```

Output:

```
64
```

Java

```
 // Java program to find number of ways to fill
// a matrix under given constraints
import java.io.*;

class Example {

    final static long mod = 100000007;

    /* Returns a raised power t under modulo mod */
    static long modPower(long a, long t, long mod)
    {
        long now = a, ret = 1;

        // Counting number of ways of filling the
        // matrix
        while (t > 0) {
            if (t % 2 == 1)
                ret = now * (ret % mod);
            now = now * (now % mod);
            t >>= 1;
        }
        return ret;
    }
    // Function calculating the answer
    static long countWays(int n, int m, int k)
    {
        // if sum of numbers of rows and columns is
        // odd i.e (n + m) % 2 == 1 and k = -1,
        // then there are 0 ways of filiing the matrix.
        if (n == 1 || m == 1)
            return 1;

        // If there is one row or one column then
        // there is only one way of filling the matrix
        else if ((n + m) % 2 == 1 && k == -1)
            return 0;

       // If the above cases are not followed then we
       // find ways to fill the n - 1 rows and m - 1
       // columns which is 2 ^ ((m-1)*(n-1)).
        return (modPower(modPower((long)2, n - 1, mod),
                                    m - 1, mod) % mod);
    }

    // Driver function for the program
    public static void main(String args[]) throws IOException
    {
        int n = 2, m = 7, k = 1;
        System.out.println(countWays(n, m, k));
    }
}
```

C#

```
 // C# program to find number of ways to fill
// a matrix under given constraints
using System;

class Example
{

    static long mod = 100000007;

    // Returns a raised power t
    // under modulo mod
    static long modPower(long a, long t,
                         long mod)
    {
        long now = a, ret = 1;

        // Counting number of ways
        // of filling the
        // matrix
        while (t > 0)
        {
            if (t % 2 == 1)
                ret = now * (ret % mod);
            now = now * (now % mod);
            t >>= 1;
        }
        return ret;
    }

    // Function calculating the answer
    static long countWays(int n, int m,
                          int k)
    {
        // if sum of numbers of rows
        // and columns is odd i.e
        // (n + m) % 2 == 1 and
        // k = -1, then there are 0
        // ways of filiing the matrix.
        if (n == 1 || m == 1)
            return 1;

        // If there is one row or one
        // column then there is only
        // one way of filling the matrix
        else if ((n + m) % 2 == 1 && k == -1)
            return 0;

        // If the above cases are not
        // followed then we find ways
        // to fill the n - 1 rows and
        // m - 1 columns which is
        // 2 ^ ((m-1)*(n-1)).
        return (modPower(modPower((long)2, n - 1,
                         mod), m - 1, mod) % mod);

    }

    // Driver Code
    public static void Main()
    {
        int n = 2, m = 7, k = 1;
        Console.WriteLine(countWays(n, m, k));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number
// of ways to fill a matrix under
// given constraints

$mod = 100000007;

// Returns a raised power t
// under modulo mod
function modPower($a, $t)
{
    global $mod;
    $now = $a; $ret = 1;

    // Counting number of ways
    // of filling the matrix
    while ($t)
    {
        if ($t & 1)
            $ret = $now * ($ret % $mod);
        $now = $now * ($now % $mod);
        $t >>= 1;
    }
    return $ret;
}

// Function calculating the answer
function countWays($n, $m, $k)
{
    global $mod;

    // if sum of numbers of rows
    // and columns is odd i.e
    // (n + m) % 2 == 1 and k = -1
    // then there are 0 ways of
    // filiing the matrix.
    if ($k == -1 and ($n + $m) % 2 == 1)
        return 0;

    // If there is one row or
    // one column then there
    // is only one way of
    // filling the matrix
    if ($n == 1 or $m == 1)
```

```
        return 1;

    // If the above cases are
    // not followed then we
    // find ways to fill the
    // n - 1 rows and m - 1
    // columns which is
    // 2 ^ ((m-1)*(n-1)).
    return (modPower(modPower(2, $n - 1),
                         $m - 1) % $mod);
}

    // Driver Code
    $n = 2;
    $m = 7;
    $k = 1;
    echo countWays($n, $m, $k);

// This code is contributed by anuj_67.
?>
```

Output:

```
64
```

The time complexity of above solution is $O(log(log n))$.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/ways-filling-matrix-product-rows-columns-equal-unity/

Chapter 132

Ways to color a 3*N board using 4 colors

Ways to color a 3*N board using 4 colors - GeeksforGeeks

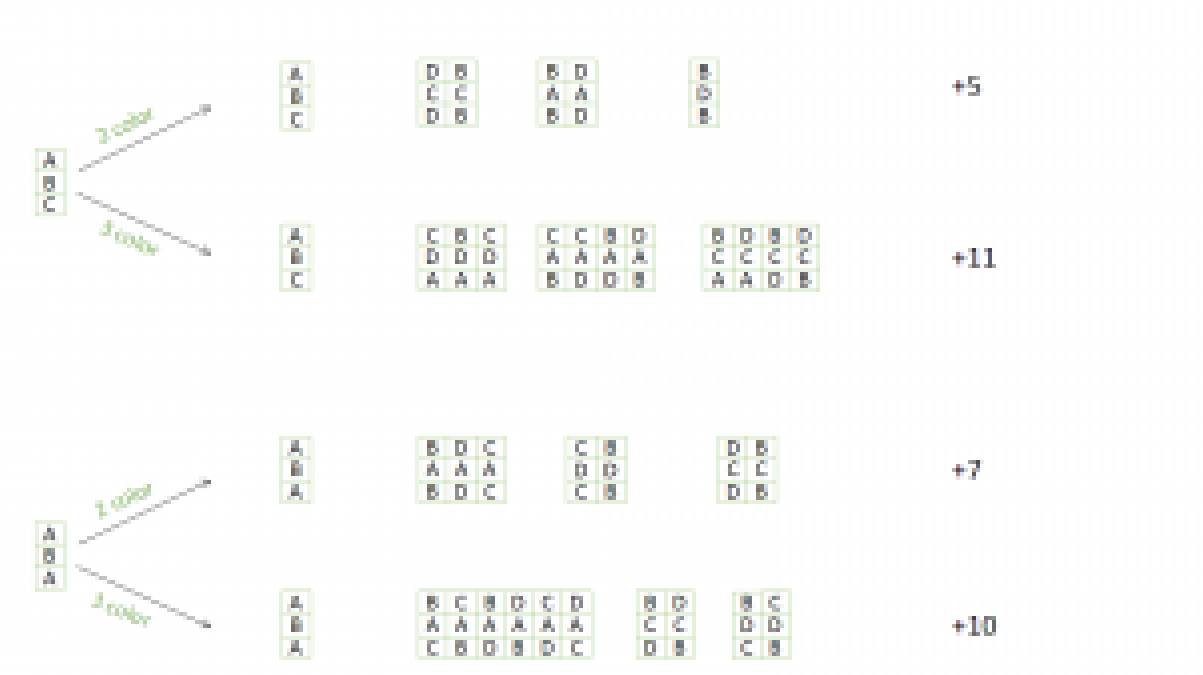
Given a 3 X n board, find the number of ways to color it using at most 4 colors such that no two adjacent boxes have the same color. Diagonal neighbors are not treated as adjacent boxes.
Output the ways%1000000007 as the answer grows quickly.

Constraints:
1<= n < 100000

Examples :

```
Input : 1
Output : 36
We can use either a combination of 3 colors
or 2 colors. Now, choosing 3 colors out of
4 is  and arranging them
in 3! ways, similarly choosing 2 colors out
of 4 is  and while arranging
we can only choose which of them could be at
centre, that would be 2 ways.
Answer = *3! +  = 36

Input : 2
Output : 588
```

We are going to solve this using dynamic approach because when a new column is added to the board, the ways in which colors are going to be filled depends just upon the color pattern in the current column. We can only have a combination of two colors and three colors in a column. All possible new columns that can be generated is given in the image. Please consider A, B, C and D as 4 colors.



All possible color combinations that can be generated from current column.

From now, we will refer 3 colors combination for a Nth column of the 3*N board as W(n) and two colors as Y(n).
We can see that each W can generate 5Y and 11W, and each Y can generate 7Y and 10W.
We get two equation from here
We have two equations now,

```
W(n+1) = 10*Y(n)+11*W(n);
Y(n+1) = 7*Y(n)+5*W(n);
```

C++

```
 // C++ program to find number of ways
// to color a 3 x n grid using 4 colors
// such that no two adjacent have same
// color
#include <iostream>
using namespace std;

int solve(int A)
{
    // When we to fill single column
    long int color3 = 24;
    long int color2 = 12;
    long int temp = 0;

    for (int i = 2; i <= A; i++)
    {
        temp = color3;
        color3 = (11 * color3 + 10 *
              color2 ) % 1000000007;

        color2 = ( 5 * temp + 7 *
              color2 ) % 1000000007;
    }

    long num = (color3 + color2)
                       % 1000000007;

    return (int)num;
}

// Driver code
int main()
{
    int num1 = 1;
    cout << solve(num1) << endl;

    int num2 = 2;
    cout << solve(num2) << endl;

    int num3 = 500;
    cout << solve(num3) << endl;

    int num4 = 10000;
    cout << solve(num4);

    return 0;
}

// This code is contributed by vt_m.
```

Java

```
 // Java program to find number of ways to color
// a 3 x n grid using 4 colors such that no two
// adjacent have same color.
public class Solution {
    public static int solve(int A) {
        long color3 = 24; // When we to fill single column
        long color2 = 12;
        long temp = 0;
        for (int i = 2; i <= A; i++)
        {
            long temp = color3;
            color3 = (11 * color3 + 10 * color2 ) % 1000000007;
            color2 = ( 5 * temp + 7 * color2 ) % 1000000007;
        }
        long num = (color3 + color2) % 1000000007;
        return (int)num;
    }

    // Driver code
    public static void main(String[] args)
    {
        int num1 = 1;
        System.out.println(solve(num1));

        int num2 = 2;
        System.out.println(solve(num2));

        int num3 = 500;
        System.out.println(solve(num3));

        int num4 = 10000;
        System.out.println(solve(num4));
    }
}
```

C#

```
 // C# program to find number of ways
// to color a 3 x n grid using 4
// colors such that no two adjacent
// have same color.
using System;

public class GFG {

    public static int solve(int A)
    {

        // When we to fill single column
        long color3 = 24;
        long color2 = 12;
        long temp = 0;
        for (int i = 2; i <= A; i++)
        {
            temp = color3;
            color3 = (11 * color3 + 10
                 * color2 ) % 1000000007;

            color2 = ( 5 * temp + 7
                 * color2 ) % 1000000007;
        }
        long num = (color3 + color2)
                            % 1000000007;
        return (int)num;
    }

    // Driver code
    public static void Main()
    {
        int num1 = 1;
        Console.WriteLine(solve(num1));

        int num2 = 2;
        Console.WriteLine(solve(num2));

        int num3 = 500;
        Console.WriteLine(solve(num3));

        int num4 = 10000;
        Console.WriteLine(solve(num4));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number of ways
// to color a 3 x n grid using 4 colors
// such that no two adjacent have same
// color
function solve($A)
{

    // When we to fill single column
    $color3 = 24;
    $color2 = 12;
    $temp = 0;
    for ($i = 2; $i <= $A; $i++)
    {
        $temp = $color3;
        $color3 = (11 * $color3 +
                   10 * $color2 ) %
                   1000000007;

        $color2 = ( 5 * $temp +
                    7 * $color2 ) %
                    1000000007;
    }

    $num = ($color3 + $color2) %
                     1000000007;

    return (int)$num;
}

// Driver code
$num1 = 1;
echo solve($num1) ,"\n";

$num2 = 2;
echo solve($num2) ,"\n";

$num3 = 500;
echo solve($num3),"\n";

$num4 = 10000;
echo solve($num4);

// This code is contributed by m_kit.
?>
```

Output :

```
36
588
178599516
540460643
```

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/ways-color-3n-board-using-4-colors/

Chapter 133

Ways to multiply n elements with an associative operation

Ways to multiply n elements with an associative operation - GeeksforGeeks

Given a number n, find the number of ways to multiply n elements with an associative operation.

Examples :

```
Input : 2
Output : 2
For a and b there are two ways to multiply them.
1. (a * b)
2. (b * a)

Input : 3
Output : 12
```

Explanation(Example 2) :

```
For a, b and c there are 12 ways to multiply them.
1.  ((a * b) * c)     2.  (a * (b * c))
3.  ((a * c) * b)     4.  (a * (c * b))
5.  ((b * a) * c)     6.  (b * (a * c))
7.  ((b * c) * a)     8.  (b * (c * a))
9.  ((c * a) * b)     10.  (c * (a * b))
11.  ((c * b) * a)    12.  (c * (b * a))
```

Approach : First, we try to find out the recurrence relation. From above examples, we can see h(1) = 1, h(2) = 2, h(3) = 12 . Now, for n elements there will be n – 1 multiplications and n – 1 parentheses. And, (a1, a2, …, an) can be obtained from (a1, a2, …, a(n – 1)) in exactly one of the two ways :

1. Take a multiplication (a1, a2, …, a(n – 1))(which has n – 2 multiplications and n – 2 parentheses) and insert the nth element 'an' on either side of either factor in one of the n – 2 multiplications. Thus, for each scheme for n – 1 numbers gives 2 * 2 * (n – 2) = 4 * (n – 2) schemes for n numbers in this way.
2. Take a multiplication scheme for (a1, a2, .., a(n-1)) and multiply on left or right by ('an'). Thus, for each each scheme for n – 1 numbers gives two schemes for n numbers in this way.

So after adding above two, we get, h(n) = (4 * n – 8 + 2) * h(n – 1), h(n) = (4 * n – 6) * h(n – 1). This recurrence relation with same initial value is satisfied by the pseudo-Catalan number. Hence, h(n) = (2 * n – 2)! / (n – 1)!

C++

```
 // CPP code to find number of ways to multiply n
// elements with an associative operation
# include <bits/stdc++.h>
using namespace std;

// Function to find the required factorial
int fact(int n)
{
    if (n == 0 || n == 1)
        return 1 ;

    int ans = 1;
    for (int i = 1 ; i <= n; i++)
        ans = ans * i ;

    return ans ;
}

// Function to find nCr
int nCr(int n, int r)
{
    int Nr = n , Dr = 1 , ans = 1;
    for (int i = 1 ; i <= r ; i++ ) {
        ans = ( ans * Nr ) / ( Dr ) ;
        Nr-- ;
        Dr++ ;
    }
    return ans ;
}

// function to find the number of ways
```

```
int solve ( int n )
{
    int N = 2*n - 2 ;
    int R = n - 1 ;
    return nCr (N, R) * fact(n - 1) ;
}

// Driver code
int main()
{
    int n = 6 ;
    cout << solve (n) ;
    return 0 ;
}
```

Java

```
 // Java code to find number of
// ways to multiply n elements
// with an associative operation
import java.io.*;

class GFG
{
// Function to find the
// required factorial
static int fact(int n)
{
    if (n == 0 || n == 1)
        return 1 ;

    int ans = 1;
    for (int i = 1 ; i <= n; i++)
        ans = ans * i ;

    return ans ;
}

// Function to find nCr
static int nCr(int n, int r)
{
    int Nr = n , Dr = 1 , ans = 1;
    for (int i = 1 ; i <= r ; i++ )
    {
        ans = ( ans * Nr ) / ( Dr ) ;
        Nr-- ;
        Dr++ ;
    }
    return ans ;
}

// function to find
// the number of ways
static int solve ( int n )
{
    int N = 2 * n - 2 ;
    int R = n - 1 ;
    return nCr (N, R) * fact(n - 1) ;
}

// Driver Code
public static void main (String[] args)
{
int n = 6 ;
System.out.println( solve (n)) ;
}
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python3 code to find number
# of ways to multiply n
# elements with an
# associative operation

# Function to find the
# required factorial
def fact(n):
    if (n == 0 or n == 1):
        return 1;

    ans = 1;
    for i in range(1, n + 1):
        ans = ans * i;

    return ans;

# Function to find nCr
def nCr(n, r):
    Nr = n ; Dr = 1 ; ans = 1;
    for i in range(1, r + 1):
        ans = int((ans * Nr) / (Dr));
        Nr = Nr - 1;
        Dr = Dr + 1;
    return ans;

# function to find
# the number of ways
def solve ( n ):
    N = 2* n - 2;
    R = n - 1 ;
    return (nCr (N, R) *
            fact(n - 1));

# Driver code
n = 6 ;
print(solve (n) );

# This code is contributed
# by mits
```

C#

```
 // C# code to find number of
// ways to multiply n elements
// with an associative operation
using System;

class GFG {

    // Function to find the
    // required factorial
    static int fact(int n)
    {
        if (n == 0 || n == 1)
            return 1 ;

        int ans = 1;
        for (int i = 1 ; i <= n; i++)
            ans = ans * i ;

        return ans ;
    }

    // Function to find nCr
    static int nCr(int n, int r)
    {
        int Nr = n , Dr = 1 , ans = 1;
        for (int i = 1 ; i <= r ; i++ )
        {
            ans = ( ans * Nr ) / ( Dr ) ;
            Nr-- ;
            Dr++ ;
        }
        return ans ;
    }

    // function to find
    // the number of ways
    static int solve ( int n )
    {
        int N = 2 * n - 2 ;
        int R = n - 1 ;
        return nCr (N, R) * fact(n - 1) ;
    }

    // Driver Code
    public static void Main ()
    {
        int n = 6 ;
        Console.WriteLine( solve (n)) ;
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP code to find number
// of ways to multiply n
// elements with an
// associative operation

// Function to find the
// required factorial
function fact($n)
{
    if ($n == 0 || $n == 1)
        return 1;

    $ans = 1;
    for ($i = 1 ; $i <= $n; $i++)
        $ans = $ans * $i;

    return $ans;
}

// Function to find nCr
function nCr($n, $r)
```

```
{
    $Nr = $n ; $Dr = 1 ; $ans = 1;
    for ($i = 1 ; $i <= $r ; $i++ )
    {
        $ans = ($ans * $Nr) /
                      ($Dr);
        $Nr--;
        $Dr++;
    }
    return $ans ;
}

// function to find
// the number of ways
function solve ( $n )
{
    $N = 2* $n - 2 ;
    $R = $n - 1 ;
    return nCr ($N, $R) *
           fact($n - 1) ;
}

// Driver code
$n = 6 ;
echo solve ($n) ;

// This code is contributed
// by ajit
?>
```

Output :

```
30240
```

Improved By : vt_m, jit_t, Mithun Kumar

Source

https://www.geeksforgeeks.org/ways-multiply-n-elements-associative-operation/

Chapter 134

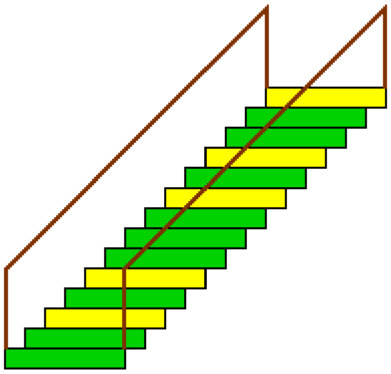
Ways to paint stairs with two colors such that two adjacent are not yellow

Ways to paint stairs with two colors such that two adjacent are not yellow - GeeksforGeeks

Given n stairs and we have 2 colour yellow and green the task is that we have to paint given stairs by given colour with condition is that we cannot paints two yellow steps directly after each other.

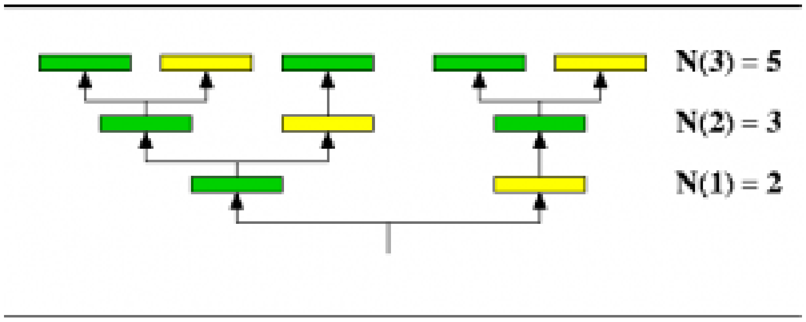
Examples :

```
Input : n = 1
Output : 2
A single stair can be colored either
as green or yellow.

Input : n = 3
Output : 5
```



Case 1: When we have 1 stair, we can paint either yellow or green.
Case 2: When we have 2 stairs, we can paint first stair by either yellow or green but for next stair we can only paint by green because we cannot paint two yellow steps directly after each other. So total cases are three YG, GG, GY.
Case 3: When we have 3 stairs then we can paint it by in 5 ways.

If we take a closer look, we can notice that it follows Fibonacci Series.



C++

```
 // C++ Program to find the number of ways to paint stairs
#include <bits/stdc++.h>
using namespace std;

// Function to find the number of ways
int ways(int n)
{
    int W[n + 1];

    // take base case for 1 and 2
    W[1] = 2;
    W[2] = 3;

    for (int i = 3; i <= n; i++)
```

```
        W[i] = W[i - 1] + W[i - 2];

    return W[n];
}

// Driven code
int main()
{
    int n = 3;
    printf("%d", ways(n));
    return 0;
}
```

Java

```
 // java Program to find the number of
// ways to paint stairs
import java.io.*;

public class GFG {

    // Function to find the number of ways
    static int ways(int n)
    {
        int []W = new int[n+1];

        // take base case for 1 and 2
        W[1] = 2;
        W[2] = 3;

        for (int i = 3; i <= n; i++)
            W[i] = W[i - 1] + W[i - 2];

        return W[n];
    }

    // Driven code
    static public void main (String[] args)
    {
        int n = 3;

        System.out.println(ways(n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Python3 code to find the number
# of ways to paint stairs

# Function to find the number of ways
def ways( n ):
    W = list()

    # take base case for 1 and 2
    W.append(0)
    W.append(2)
    W.append(3)

    i = 3
    while i <= n:
        W.append(W[i - 1] + W[i - 2])
        i = i + 1

    return W[n]

# Driver code
n = 3
print(ways(n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# Program to find the number of
// ways to paint stairs
using System;

public class GFG {

    // Function to find the number of ways
    static int ways(int n)
    {
        int []W =new int[n+1];

        // take base case for 1 and 2
        W[1] = 2;
        W[2] = 3;

        for (int i = 3; i <= n; i++)
            W[i] = W[i - 1] + W[i - 2];

        return W[n];
    }
    // Driven code
    static public void Main ()
    {
        int n = 3;

        Console.WriteLine(ways(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the
// number of ways to paint stairs

// Function to find the
// number of ways
function ways($n)
{

    // take base case
    // for 1 and 2
    $W[1] = 2;
    $W[2] = 3;

    for ($i = 3; $i <= $n; $i++)
        $W[$i] = $W[$i - 1] +
                 $W[$i - 2];

    return $W[$n];
}

// Driven code
$n = 3;
echo ways($n);

// This code is contributed by ajit
?>
```

Output :

```
5
```

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

We can solve this problem in O(Log n) time also using matrix exponentiation solution for n-th Fibonacci Number.

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/ways-paint-stairs-two-colors-two-adjacent-not-yellow/

Chapter 135

Ways to represent a number as a sum of 1's and 2's

Ways to represent a number as a sum of 1's and 2's - GeeksforGeeks

Given a positive integer **N**. The task is to find the number of ways of representing N as a sum of 1s and 2s.

Examples:

```
Input : N = 3
Output : 3
3 can be represented as (1+1+1), (2+1), (1+2).

Input : N = 5
Output : 8
```

For N = 1, answer is 1.
For N = 2. (1 + 1), (2), answer is 2.
For N = 3. (1 + 1 + 1), (2 + 1), (1 + 2), answer is 3.
For N = 4. (1 + 1 + 1 + 1), (2 + 1 + 1), (1 + 2 + 1), (1 + 1 + 2), (2 + 2) answer is 5.
And so on.

It can be observe that it form Fibonacci Series. So, the number of ways of representing N as a sum of 1s and 2s is $(N + 1)^{th}$ Fibonacci number.
How ?
We can easily see that the recursive function is exactly same as Fibonacci Numbers. To obtain the sum of N, we can add 1 to N – 1. Also, we can add 2 to N – 2. And only 1 and 2 are allowed to make the sum N. So, to obtain sum N using 1s and 2s, total ways are: number of ways to obtain (N – 1) + number of ways to obtain (N – 2).

We can find N'th Fibonacci Number in O(Log n) time. Please refer method 5 of this post.

Below is C++ implementation of this approach:

```
 // C++ program to find number of ways to representing
// a number as a sum of 1's and 2's
#include <bits/stdc++.h>
using namespace std;

// Function to multiply matrix.
void multiply(int F[2][2], int M[2][2])
{
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

// Power function in log n
void power(int F[2][2], int n)
{
    if( n == 0 || n == 1)
        return;
    int M[2][2] = {{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);
}

/* function that returns (n+1)th Fibonacci number
   Or number of ways to represent n as sum of 1's
   2's */
int countWays(int n)
{
    int F[2][2] = {{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n);
    return F[0][0];
}

// Driver program
int main()
{
    int n = 5;
    cout << countWays(n) << endl;
    return 0;
}
```

Output:

```
8
```

Time Complexity: O(logn).

Source

https://www.geeksforgeeks.org/ways-to-represent-a-number-as-a-sum-of-1s-and-2s/

Chapter 136

Ways to select one or more pairs from two different sets

Ways to select one or more pairs from two different sets - GeeksforGeeks

Given two positive numbers 'n' and 'm' (n <= m) which represent total number of items of first and second type of sets respectively. Find total number of ways to select at-least one pair by picking one item from first type(I) and another item from second type(II). In any arrangement, an item should not be common between any two pairs.
Note: Since answer can be large, output it in modulo **1000000007**.

```
Input: 2 2
Output: 6
Explanation
Let's denote the items of I type
as a, b and II type as c, d i.e,
Type I -  a, b
Type II - c, d
Ways to arrange one pair at a time
1. a --- c
2. a --- d
3. b --- c
4. b --- d
Ways to arrange two pairs at a time
5. a --- c, b --- d
6. a --- d, b --- c

Input: 2 3
Output: 12

Input: 1 2
Output: 2
```

The approach is simple, we only need the combination of choosing '**i**' items from '**n**' type and '**i**' items from '**m**' type and multiply them(Rule of product) where '**i**' varies from **1** to '**n**'. But we can also permute the resultant product in 'i' ways therefore we need to multiply with **i!**. After that take the sum(Rule of sum) of all resultant product to get the final answer.

C++

```
 // C++ program to find total no. of ways
// to form a pair in two different set
#include <bits/stdc++.h>
using namespace std;

// initialize global variable so that
// it can access by preCalculate() and
// nCr() function
int* fact, *inverseMod;
const int mod = 1e9 + 7;

/* Iterative Function to calculate (x^y)%p in O(log y) */
int power(int x, int y, int p)
{
    int res = 1; // Initialize result

    x = x % p; // Update x if it is more than or
               // equal to p

    while (y) {

        // If y is odd, multiply x with result
        if (y & 1)
            res = (1LL * res * x) % p;

        // y must be even now
        y = y >> 1; // y = y/2
        x = (1LL * x * x) % p;
    } // trace(res);

    return res;
}

// Pre-calculate factorial and
// Inverse of number
void preCalculate(int n)
{
    fact[0] = inverseMod[0] = 1;
    for (int i = 1; i <= n; ++i) {

        fact[i] = (1LL * fact[i - 1] * i) % mod;
        inverseMod[i] = power(fact[i], mod - 2, mod);
    }
}

// utility function to calculate nCr
int nPr(int a, int b)
{
    return (1LL * fact[a] * inverseMod[a - b]) % mod;
}

int countWays(int n, int m)
{
    fact = new int[m + 1];
    inverseMod = new int[m + 1];

    // Pre-calculate factorial and
    // inverse of number
    preCalculate(m);

    // Initialize answer
    int ans = 0;
    for (int i = 1; i <= n; ++i) {

        ans += (1LL * ((1LL * nPr(n, i)
                * nPr(m, i)) % mod)
                * inverseMod[i]) % mod;
        if (ans >= mod)
            ans %= mod;
    }

    return ans;
}

// Driver program
int main()
{
    int n = 2, m = 2;

    cout << countWays(n, m);

    return 0;
}
```

Java

```
 // Java program to find total
// no. of ways to form a pair
// in two different set

public class Test
{
    static long[] fact;
    static long[] inverseMod;
    static int mod=1000000007;

    /* Iterative Function to calculate
       (x^y)%p in O(log y) */

    static long power(long x, int y, int p)
    {
        // Initialize result
        long res = 1;

        // Update x if it is more than or
        // equal to p
        x = x % p;

        while (y!=0)
        {

            // If y is odd, multiply
            // x with result
            if ((y & 1)!=0)
                res = (1 * res * x) % p;
            }
            // y must be even now
            y = y >> 1;

            x = (1 * x * x) % p;
        }


        return res;
    }

    // Pre-calculate factorial and
    // Inverse of number
    public static void preCalculate(int n)
    {
        //int fact[]=new long[n];
        // int inverseMod[]=new long[n];
        fact[0] = 1;
        inverseMod[0] = 1;

        for (int i = 1; i <= n; i++)
        {

                fact[i] = (1 * fact[i - 1] * i)
                                          % mod;

                inverseMod[i] = power(fact[i],
                                    mod - 2, mod);

        }
    }

    // utility function to calculate nCr
    public static long nPr(int a, int b)
    {

        return (1 * fact[a] * inverseMod[a - b])
                                    % (long)mod;
    }

    public static int countWays(int n, int m)
    {

        fact = new long[m + 1];
        inverseMod = new long[m + 1];

        // Pre-calculate factorial and
        // inverse of number
        preCalculate(m);

        // Initialize answer
        long ans = 0;

        for (int i = 1; i <= n; i++) {

            ans = ans+(1 * ((1 * nPr(n, i)*
                            nPr(m, i)) % mod)*
                            (inverseMod[i])) % mod;

            if (ans >= mod)
                ans %= mod;
        }
        return (int)ans;
     }

    /* Driver program */
    public static void main(String[] args)
    {
        int n = 2, m = 2;

        System.out.println(countWays(n, m));
    }
}

// This code is contributed by Gitanjali
```

Output:

```
6
```

Time complexity: O(m*log(mod))
Auxiliary space: O(m)

Source

https://www.geeksforgeeks.org/ways-to-select-one-or-more-pairs-from-two-different-sets/

Chapter 137

Ways to sum to N using array elements with repetition allowed

Ways to sum to N using array elements with repetition allowed - GeeksforGeeks

Given a set of m distinct positive integers and a value 'N'. The problem is to count the total number of ways we can form 'N' by doing sum of the array elements. Repetitions and different arrangements are allowed.

Examples :

```
Input : arr = {1, 5, 6}, N = 7
Output : 6

Explanation:- The different ways are:
1+1+1+1+1+1+1
1+1+5
1+5+1
5+1+1
1+6
6+1

Input : arr = {12, 3, 1, 9}, N = 14
Output : 150
```

Approach: The approach is based on the concept of dynamic programming.

```
countWays(arr, m, N)
        Declare and initialize count[N + 1] = {0}
        count[0] = 1
        for i = 1 to N
```

```
            for j = 0 to m - 1
                if i >= arr[j]
                   count[i] += count[i - arr[j]]
        return count[N]
```

C++

```
 // C++ implementation to count ways
// to sum up to a given value N
#include <bits/stdc++.h>

using namespace std;

// function to count the total
// number of ways to sum up to 'N'
int countWays(int arr[], int m, int N)
{
    int count[N + 1];
    memset(count, 0, sizeof(count));

    // base case
    count[0] = 1;

    // count ways for all values up
    // to 'N' and store the result
    for (int i = 1; i <= N; i++)
        for (int j = 0; j < m; j++)

            // if i >= arr[j] then
            // accumulate count for value 'i' as
            // ways to form value 'i-arr[j]'
            if (i >= arr[j])
                count[i] += count[i - arr[j]];

    // required number of ways
    return count[N];

}

// Driver code
int main()
{
    int arr[] = {1, 5, 6};
    int m = sizeof(arr) / sizeof(arr[0]);
    int N = 7;
    cout << "Total number of ways = "
        << countWays(arr, m, N);
    return 0;
}
```

Java

```
 // Java implementation to count ways
// to sum up to a given value N

class Gfg
{
    static int arr[] = {1, 5, 6};

    // method to count the total number
    // of ways to sum up to 'N'
    static int countWays(int N)
    {
        int count[] = new int[N + 1];

        // base case
        count[0] = 1;

        // count ways for all values up
        // to 'N' and store the result
        for (int i = 1; i <= N; i++)
            for (int j = 0; j < arr.length; j++)

                // if i >= arr[j] then
                // accumulate count for value 'i' as
                // ways to form value 'i-arr[j]'
                if (i >= arr[j])
                    count[i] += count[i - arr[j]];

        // required number of ways
        return count[N];

    }

    // Driver code
    public static void main(String[] args)
    {
        int N = 7;
        System.out.println("Total number of ways = "
                                + countWays(N));
    }
}
```

Python3

```python
# Python3 implementation to count
# ways to sum up to a given value N

# Function to count the total
# number of ways to sum up to 'N'
def countWays(arr, m, N):

    count = [0 for i in range(N + 1)]

    # base case
    count[0] = 1

    # Count ways for all values up
    # to 'N' and store the result
    for i in range(1, N + 1):
        for j in range(m):

            # if i >= arr[j] then
            # accumulate count for value 'i' as
            # ways to form value 'i-arr[j]'
            if (i >= arr[j]):
                count[i] += count[i - arr[j]]

    # required number of ways
    return count[N]

# Driver Code
arr = [1, 5, 6]
m = len(arr)
N = 7
print("Total number of ways = ",
           countWays(arr, m, N))

# This code is contributed by Anant Agarwal.
```

C#

```csharp
 // C# implementation to count ways
// to sum up to a given value N
using System;

class Gfg
{
    static int []arr = {1, 5, 6};

    // method to count the total number
    // of ways to sum up to 'N'
    static int countWays(int N)
    {
        int []count = new int[N+1];

        // base case
        count[0] = 1;

        // count ways for all values up
        // to 'N' and store the result
        for (int i = 1; i <= N; i++)
            for (int j = 0; j < arr.Length; j++)

                // if i >= arr[j] then
                // accumulate count for value 'i' as
                // ways to form value 'i-arr[j]'
                if (i >= arr[j])
                    count[i] += count[i - arr[j]];

        // required number of ways
        return count[N];

    }

    // Driver code
    public static void Main()
    {
        int N = 7;
        Console.Write("Total number of ways = "
                                    + countWays(N));
    }
}

//This code is contributed by nitin mittal.
```

Output:

```
Total number of ways = 6
```

Time Complexity: O(N*m)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/ways-sum-n-using-array-elements-repetition-allowed/

Chapter 138

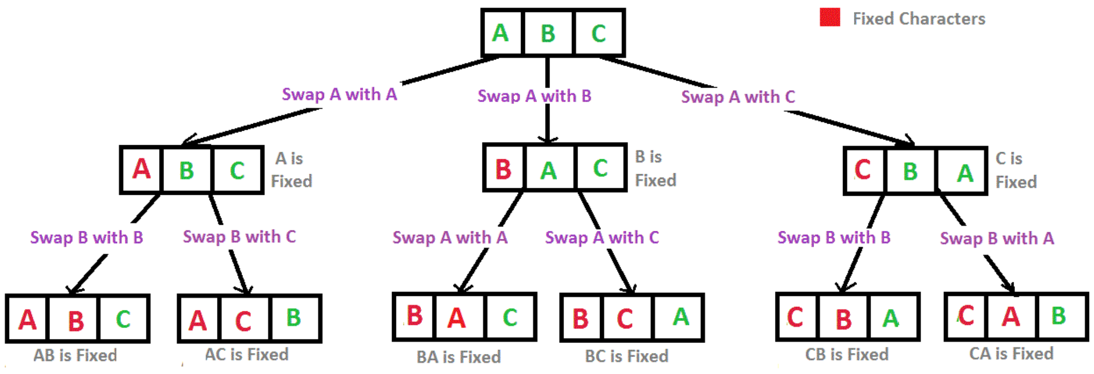
Write a program to print all permutations of a given string

Write a program to print all permutations of a given string - GeeksforGeeks

A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! permutation.
Source: Mathword(http://mathworld.wolfram.com/Permutation.html)

Below are the permutations of string ABC.
ABC ACB BAC BCA CBA CAB

Here is a solution that is used as a basis in backtracking.



Recursion Tree for Permutations of String "ABC"

C/C++

```
 // C program to print all permutations with duplicates allowed
#include <stdio.h>
```

```
#include <string.h>

/* Function to swap values at two pointers */
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

/* Function to print permutations of string
   This function takes three parameters:
   1. String
   2. Starting index of the string
   3. Ending index of the string. */
void permute(char *a, int l, int r)
{
   int i;
   if (l == r)
     printf("%s\n", a);
   else
   {
       for (i = l; i <= r; i++)
       {
          swap((a+l), (a+i));
          permute(a, l+1, r);
          swap((a+l), (a+i)); //backtrack
       }
   }
}

/* Driver program to test above functions */
int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}
```

Java

```
 // Java program to print all permutations of a
// given string.
public class Permutation
{
    public static void main(String[] args)
    {
        String str = "ABC";
        int n = str.length();
        Permutation permutation = new Permutation();
        permutation.permute(str, 0, n-1);
    }

    /**
     * permutation function
     * @param str string to calculate permutation for
     * @param l starting index
     * @param r end index
     */
    private void permute(String str, int l, int r)
    {
        if (l == r)
            System.out.println(str);
        else
        {
            for (int i = l; i <= r; i++)
            {
                str = swap(str,l,i);
                permute(str, l+1, r);
                str = swap(str,l,i);
            }
        }
    }

    /**
     * Swap Characters at position
     * @param a string value
     * @param i position 1
     * @param j position 2
     * @return swapped string
     */
    public String swap(String a, int i, int j)
    {
        char temp;
        char[] charArray = a.toCharArray();
        temp = charArray[i] ;
        charArray[i] = charArray[j];
        charArray[j] = temp;
        return String.valueOf(charArray);
    }

}

// This code is contributed by Mihir Joshi
```

Python

```
 # Python program to print all permutations with
# duplicates allowed

def toString(List):
    return ''.join(List)

# Function to print permutations of string
# This function takes three parameters:
# 1. String
# 2. Starting index of the string
# 3. Ending index of the string.
def permute(a, l, r):
    if l==r:
        print toString(a)
    else:
        for i in xrange(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n-1)

# This code is contributed by Bhavya Jain
```

C#

```
 // C# program to print all
// permutations of a given string.
using System;

class GFG
{
    /**
    * permutation function
    * @param str string to
       calculate permutation for
    * @param l starting index
    * @param r end index
    */
    private static void permute(String str,
                                int l, int r)
    {
        if (l == r)
            Console.WriteLine(str);
        else
        {
            for (int i = l; i <= r; i++)
            {
                str = swap(str, l, i);
                permute(str, l + 1, r);
                str = swap(str, l, i);
            }
        }
    }

    /**
    * Swap Characters at position
    * @param a string value
    * @param i position 1
    * @param j position 2
    * @return swapped string
    */
    public static String swap(String a,
                              int i, int j)
    {
        char temp;
        char[] charArray = a.ToCharArray();
        temp = charArray[i] ;
        charArray[i] = charArray[j];
        charArray[j] = temp;
        string s = new string(charArray);
        return s;
    }

// Driver Code
public static void Main()
{
    String str = "ABC";
    int n = str.Length;
    permute(str, 0, n-1);
}
}

// This code is contributed by mits
```

PHP

```
 <?php
// PHP program to print all
// permutations of a given string.


/**
* permutation function
* @param str string to
*  calculate permutation for
* @param l starting index
* @param r end index
*/
function permute($str, $l, $r)
{
    if ($l == $r)
        echo $str. "\n";
    else
    {
        for ($i = $l; $i <= $r; $i++)
        {
            $str = swap($str, $l, $i);
            permute($str, $l + 1, $r);
            $str = swap($str, $l, $i);
        }
    }
}

/**
* Swap Characters at position
* @param a string value
* @param i position 1
* @param j position 2
* @return swapped string
*/
function swap($a, $i, $j)
{
    $temp;
    $charArray = str_split($a);
    $temp = $charArray[$i] ;
    $charArray[$i] = $charArray[$j];
    $charArray[$j] = $temp;
    return implode($charArray);
}

// Driver Code
$str = "ABC";
$n = strlen($str);
permute($str, 0, $n - 1);

// This code is contributed by mits.
```

```
?>
```

Output:

```
ABC
ACB
BAC
BCA
CBA
CAB
```

Algorithm Paradigm: Backtracking
Time Complexity: O(n*n!) Note that there are n! permutations and it requires O(n) time to print a a permutation.

Note : The above solution prints duplicate permutations if there are repeating characters in input string. Please see below link for a solution that prints only distinct permutations even if there are duplicates in input.

Print all distinct permutations of a given string with duplicates.
Permutations of a given string using STL

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/

Chapter 139

itertools.combinations() module in Python to print all possible combinations

itertools.combinations() module in Python to print all possible combinations - GeeksforGeeks

Given an array of size n, generate and print all possible combinations of r elements in array.

Examples:

```
Input : arr[] = [1, 2, 3, 4],
            r = 2
Output : [[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]
```

This problem has existing recursive solution please refer Print all possible combinations of r elements in a given array of size n link. We will solve this problem in python using **itertools.combinations()** module.

It returns r length subsequences of elements from the input iterable. Combinations are emitted in lexicographic sort order. So, if the input iterable is sorted, the combination tuples will be produced in sorted order.

- **itertools.combinations(iterable, r) :**
  It return r-length tuples in sorted order with no repeated elements. For Example, combinations('ABCD', 2) ==> [AB, AC, AD, BC, BD, CD].
- **itertools.combinations_with_replacement(iterable, r) :**
  It return r-length tuples in sorted order with repeated elements. For Example, combinations_with_replacement('ABCD', 2) ==> [AA, AB, AC, AD, BB, BC, BD, CC, CD, DD].

```
[[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]
```

Source

https://www.geeksforgeeks.org/itertools-combinations-module-python-print-possible-combinations/