COMBINATORICS

What is combinatorics: Combinatorics is a branch of mathematics primarily concerned with counting and certain properties of finite structures.

It deals with the study of enumeration, combination, and permutation of sets of elements and the mathematical relations that characterize their properties.

In other word we can also that Combinatorics is all about number of ways of choosing some objects out of a collection and/or number of ways of their arrangement. For example suppose there are five members in a club, let's say there names are A, B, C, D, and E, and one of them is to be chosen as the coordinator. Clearly any one out of them can be chosen so there are 5 ways. Now suppose two members are to be chosen for the position of coordinator and co-coordinator. Now, we can choose A as coordinator and one out of the rest 4 as co-coordinator. Similarly we can choose B as coordinator and one of out the remaining 4 as co-coordinator, and similarly with C, D and E. So there will be total 20 possible ways.

Basic Combinatorics Rules:

Suppose there are two sets A and B.

The basic rules of combinatorics one must remember are:

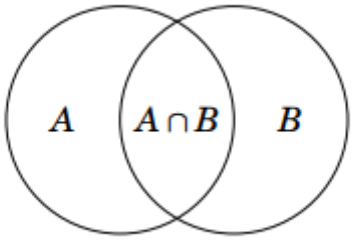
1. **The Rule of Product**:
   The product rule states that if there are X number of ways to choose one element from A and Y number of ways to choose one element from B, then there will be X×Y number of ways to choose two elements, one from A and one from B.

2. **The Rule of Sum**:
   The sum rule states that if there are X number of ways to choose one element from A and Y number of ways to choose one element from B, then there will be X+Y number of ways to choose one element that can belong to either A or to B.

3. **Rule of Inclusion-Exclusion :**

   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 any set A and B,

|A U B| = |A| + |B| - |A ∩ B|.

These rules can be used for a finite collection of sets.

Examples:

Example 1 – In how many ways can 3 winning prizes be given to the top 3 players in a game played by 12 players?

Solution – We have to distribute 3 prizes among 12 players. This task can be divided into 3 subtasks of assigning a single prize to a certain player.

Giving out the first prize can be done in 12 different ways. After giving out the first prize, two prizes remain and 11 players remain. Similarly, the second prize and third prize can be given in 11 ways and 10 ways. The total number of ways by the product rule is 12 * 11 * 10 = 1320.

Example 2 – How many distinct license plates are possible in the given format- Two alphabets in uppercase, followed by two digits then a hyphen, and finally four digits. Sample- AB12-3456.

Solution – There are 26 possibilities for each of the two letters and 10 possibilities for each of the digits. Therefore, the total number of possibilities is – 26 * 26 * 10 * 10 * 10 * 10 * 10 * 10 = 676000000.

Example 3 – How many variable names of length up to 3 exist if the variable names are alphanumeric and case-sensitive with the restriction that the first character has to be an alphabet?

Solution – Let N1, N2, and N3 denote the number of possible variable names of lengths 1, 2, and 3. Therefore, the total number of variable names is N1 + N2 + N3.

For N1, there are only 52 possibilities since the first character has to be an alphabet.

For N2, there are 52 * 62 = 3224 possibilities.

For N3, there are 52 * 62 * 62 = 199888 possibilities.

Therefore, total number of variable names = 52 + 3224 + 199888 = 203164

Example 4 – 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.

Factorials:

Factorial of a number n is defined as a product of all positive descending integers, Factorial of n is denoted by n!. Factorial can be calculated using the following recursive formula where the recursive call is made to a multiplicity of all the numbers lesser than the number for which the factorial is computed as the formula to calculate factorial is as follows:

n! = n * [(n-1)!]

i.e factorial of n - (n!) = n * (n-1) * .....* 3 * 2* 1.

Note: Factorial of 0 is 1.

1. Factorial Program using Loop:

```
#include <iostream>
using namespace std;
int main()
{
   int i,fact=1,number;
  cout<<"Enter any Number: ";
 cin>>number;
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  cout<<"Factorial of " <<number<<" is: "<<fact<<endl;
  return 0;
}
```

2. Factorial Program using Recursion:

```
#include<iostream>
using namespace std;
int main()
{
int factorial(int);
int fact,value;
cout<<"Enter any number: ";
cin>>value;
fact=factorial(value);
cout<<"Factorial of a number is: "<<fact<<endl;
return 0;
}
int factorial(int n)
{
if(n<0)
return(-1); /*Wrong value*/
if(n==0)
return(1);  /*Terminating condition*/
else
{
return(n*factorial(n-1));
}
}
```

Permutation

Permutation is defined as a mathematical calculation that tells us the way a particular set of data are arranged in a particular way. In simple word we can say that permutation is the number of ways objects can be ordered and arranged.

If n objects are available and we arrange all, then every arrangement possible is called a permutation. In other words, arranging some objects in a specific order is called permutation. For example, we need to choose two numbers out of {1, 2, 3}, then there can be 6 ways in which we can do the same. (1, 2), (2, 1), (2, 3), (3, 2), (1, 3), and (3, 1). As we can see here both (1, 2) and (2, 1) are two different permutations, unlike combinations where (1, 2) and (2, 1) are the same combination.

Representation of Permutation:

- P(n, r)
- nPr
- Pn,k

Permutation Formula (nPr):

$$P(n, r) = n! / (n-r)!$$

where,

- **n** is the Number of Total Objects
- **r** is the Number of Objects Chosen at Once
- $0 \leq r \leq n$

Types of Permutation:

In the study of permutation, there are some cases such as:

- Permutation with Repetition
- Permutation without Repetition
- Permutation of Multi-Sets

- **Permutation With Repetition:**

  This is the simplest of the lot. In such problems, the objects can be repeated. Let's understand these problems with some examples.

  Example: How many 3-digit numbers greater than 500 can be formed using 3, 4, 5, and 7?

  Since a three-digit number, greater than 500 will have either 5 or 7 at its hundredth place, we have 2 choices for this place.

  There is no restriction on repetition of the digits, hence for the remaining 2 digits we have 4 choices each

  So the total permutations are,

  $2 \times 4 \times 4 = 32$

- **Permutation Without Repetition:**

  In this class of problems, the repetition of objects is not allowed. Let's understand these problems with some examples.

  Example: How many 3-digit numbers divisible by 3 can be formed using digits 2, 4, 6, and 8 without repetition?

  For a number to be divisible by 3, the sum of it digits must be divisible by 3.

  From the given set, various arrangements like 444 can be formed but since repetition isn't allowed we won't be considering them.

  We are left with just 2 cases i.e. 2, 4, 6 and 4, 6, 8

  Number of arrangements are 3! in each case

  Hence the total number of permutations are: $3! + 3! = 12$

- Permutation of Multi-Sets:

  Permutation when the objects are not distinct

  This can be thought of as the distribution of n objects into r boxes where the repetition of objects is allowed and any box can hold any number of objects.

  1st box can hold n objects

  2nd box can hold n objects

  3rd box can hold n objects

  . .

  . .

  . .

  rth box can hold n objects

  Hence total number of arrangements are,

  $n \times n \times n \ldots$ (r times) $= nr$

  Example: A police officer visits the crime scene 3 times a week for investigation. Find the number of ways to schedule his visit if there is no restriction on the number of visits per day?

  Number of ways to schedule first visit is 7 (any of the 7 days)

  Number of ways to schedule second visit is 7 (any of the 7 days)

  Number of ways to schedule third visit is 7 (any of the 7 days)

  Hence, the number of ways to schedule first and second and third visit is

  $7 \times 7 \times 7 = 7^3 = 343$.

Programs for permutations coefficient:

1. Using the factorial:

```cpp
#include<bits/stdc++.h>
using namespace std;
int permutationCoeff(int n, int k)
{
    int fact[n + 1];
    fact[0] = 1;
    for(int i = 1; i <= n; i++)
    fact[i] = i * fact[i - 1];
    return fact[n] / fact[n - k];
}

int main()
{
    int n = 10, k = 2;

    cout << "Value of P(" << n << ", "
        << k << ") is "
        << permutationCoeff(n, k);

    return 0;
}

```

2. Just using the formula:

```cpp
#include <iostream>
using namespace std;

int PermutationCoeff(int n, int k)
{
    int P = 1;

    // Compute n*(n-1)*(n-2)....(n-k+1)
    for (int i = 0; i < k; i++)
        P *= (n-i) ;

    return P;
}

int main()
{
    int n = 10, k = 2;
    cout << "Value of P(" << n << ", " << k << ") is " <<
        PermutationCoeff(n, k);

    return 0;
}

```

3. Using dynamic programming:

```cpp
#include <bits/stdc++.h>
using namespace std;
int permutationCoeff(int n, int k)
{
    int P[n + 1][k + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= std::min(i, k); j++)
        {
            if (j == 0)
                P[i][j] = 1;
            else
                P[i][j] = P[i - 1][j] + (j * P[i - 1][j - 1]);
            P[i][j + 1] = 0;
        }
    }
    return P[n][k];
}

int main()
{
    int n = 10, k = 2;
    cout << "Value of P(" << n <<" " << k<< ") is " <<
        permutationCoeff(n, k);

    return 0;
}

```

Combination

Combination is a way of choosing items from a set, such as (unlike permutations) the order of selection doesn't matter. In smaller cases, it's possible to count the number of combinations. Combination refers to the mixture of n things taken k at a time without repetition. To know the combinations in the case where repetition is allowed, the terms like k-selection or k-combination along with repetition are often used. For instance, if we've two elements A and B, then there's just one way to select two items, we select both of them.

Combination is the choice of selecting r things from a group of n things without replacement and where the order of selection is not important.

Number of combinations when 'r' elements are selected out of a complete set of 'n' elements is denoted by nCr.

Representation of Combination:

- C(n, r)
- nCr
- Cn,k

Combination Formula (nCr):

$$C(n, r) = n! / ((n-r)! * r!)$$

where,

- **n** is the Number of Total Objects
- **r** is the Number of Objects Chosen at Once
- $0 \leq r \leq n$

Example:

1. Handshaking problem:
   Handshaking problem is one of the most interesting problems in combination. It is used to find that in a room full of people how many handshakes are required for everybody to shake everybody else's hand exactly once.

Basically when there are 2 people there will be one handshake and if there are three people there will be 3 handshakes and so on. This many people we can count but let's suppose there are thousands of people in a hall then we can't count each handshake here the need for the combination arises.

Number of People	Possible Combinations	Minimum Handshake required
Two People	A-B	1 handshake
Three People	A-B A-C B-C	3 handshake
Four People	A-B A-C A-D B-C B-D C-D	6 handshake

To see the people present, and consider one person at a time. The first person will shake hands with n – 1 other people. The next person will shake hands with n-2 other people, not counting the first person again. Following this, it will give us a total number of
(n – 1) + (n – 2) + … + 2 + 1
= n(n – 1)/ 2 handshakes.

- ***Total Number of Handshakes = n × (n – 1)/2***
- ***Total Number of Handshakes = nC2***

Properties of combination coefficient:

1. ${}^{n}Cr = {}^{n}Cn\text{-}r$.
2. ${}^{n}Cx = {}^{n}Cy$, then either $x = y$ or $x + y = n$.
3. Recursive formula for combination:

   ${}^{n-1}Cr + {}^{n-1}Cr_{-1} = {}^{n}Cr$
4. ${}^{n}Cr + {}^{n}Cr_{+1} = {}^{n+1}Cr_{+1}$

Programs for combination coefficient:

1. Using factorial function:

```cpp
#include <iostream>
using namespace std;
int fact(int n) {
    if (n == 0 || n == 1)
    return 1;
    else
    return n * fact(n - 1);
}
int main() {
    int n, r, result;
    cout<<"Enter n : ";
    cin>>n;
    cout<<"\nEnter r : ";
    cin>>r;
    result = fact(n) / (fact(r) * fact(n-r));
    cout << "\nThe result : " << result;
    return 0;
}
```

2. Using formula directly:

```cpp
#include <iostream>
using namespace std;
int fact(int n){
    if(n==0) return 1;
    if (n>0) return n*fact(n-1);
};

int NCR(int n,int r){
    if(n==r) return 1;
    if (r==0&&n!=0) return 1;
    else return (n*fact(n-1))/fact(n-1)*fact(n-r);
};

int main(){
    int n;  //cout<<"Enter A Digit for n";
    cin>>n;
    int r;
        //cout<<"Enter A Digit for r";
    cin>>r;
    int result=NCR(n,r);
    cout<<result;
    return 0;
}
```

Binomial coefficient

A binomial coefficient C(n, k) can be defined as the coefficient of x^k in the expansion of (1 + x)^n.

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 a n-element set.

$(1+x)^2 = 1 + 2x + x^2 = {}^2C_0 + {}^2C_1\,x + {}^2C_2\,x^2$

$(1+x)^3 = 1 + 3x + 3x^2 + x^3 = {}^3C_0 + {}^3C_1\,x + {}^3C_2\,x^2 + {}^3C_3\,x^3$

..

..

So on.

Thus, we can see that combinations are only the binomial coefficients.

Example:

Consider expanding $(x+2)^5$:

$(x+2)^5=(x+2)(x+2)(x+2)(x+2)(x+2)$

One quickly realizes that this is a very tedious calculation involving multiple applications of the distributive property. The binomial theorem provides a method of expanding binomials raised to powers without directly multiplying each factor:

$(x+y)^n = {}^nC_0\, x^n\, y^0 + {}^nC_1\, x^{n-1}\, y^1 + {}^nC_2\, x^{n-2}\, y^2 + \ldots\ldots\ldots\ldots + {}^nC_{n-1}\, x^1\, y^{n-1} + {}^nC_n\, x^0\, y^n$

More compactly we can write,

$(x+y)^n = \sum_{k=0}^{n} {}^nC_k\, x^{n-k}\, y^k$

Program to find binomial coefficient:

1. **Recursively:**
   The formula for recursively solving binomial coefficients.

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

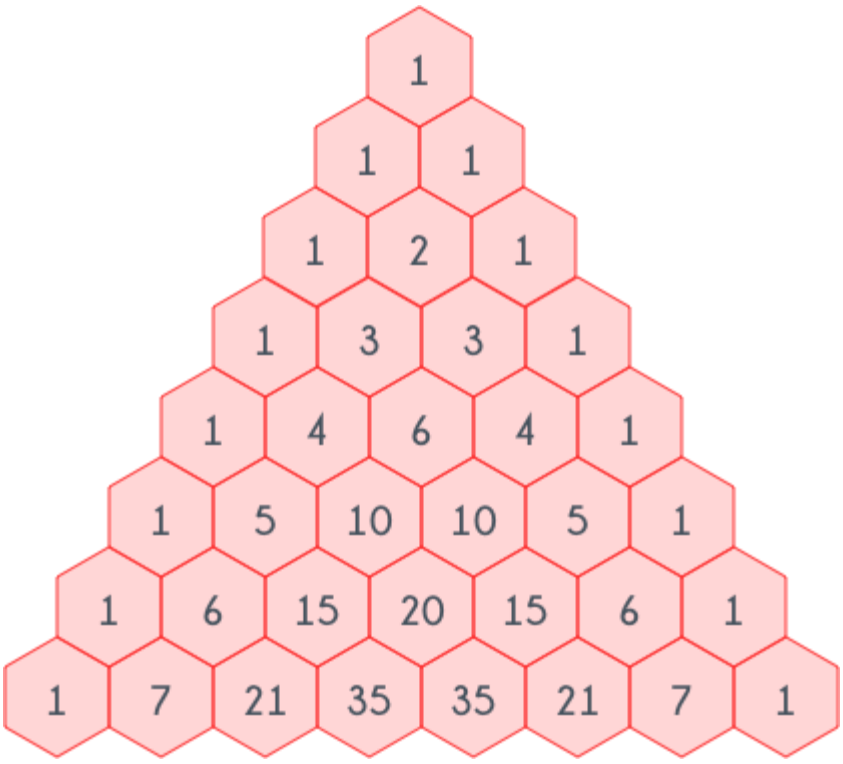
   C(n, 0) = C(n, n) = 1

```cpp
#include <bits/stdc++.h>
using namespace std;
int binomialCoeff(int n, int k)
{
    if (k > n)
        return 0;
    if (k == 0 || k == n)
        return 1;
    return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k
        );
}

int main()
{
    int n = 5, k = 2;
    cout << "Value of C(" << n << ", " << k << ") is "
        << binomialCoeff(n, k);
    return 0;
}
```

2. Using factorial by direct formula:
   ${}^{n}Cr = n! / ((n-r)! * r!)$

```cpp
#include <iostream>
unsigned long long factorial(int n) {
    unsigned long long result = 1;
    for (int i = 2; i <= n; ++i) {
        result *= i;
    }
    return result;
}
unsigned long long binomialCoefficient(int n, int k) {
    if (k > n) {
        return 0;
    }
    return factorial(n) / (factorial(k) * factorial(n - k));
}

int main() {
    int n = 5; // Example value for n
    int k = 2; // Example value for k
    std::cout << "Binomial coefficient of " << n << " choose "
        << k << " is: " << binomialCoefficient(n, k) << std
        ::endl;
    return 0;
}
```

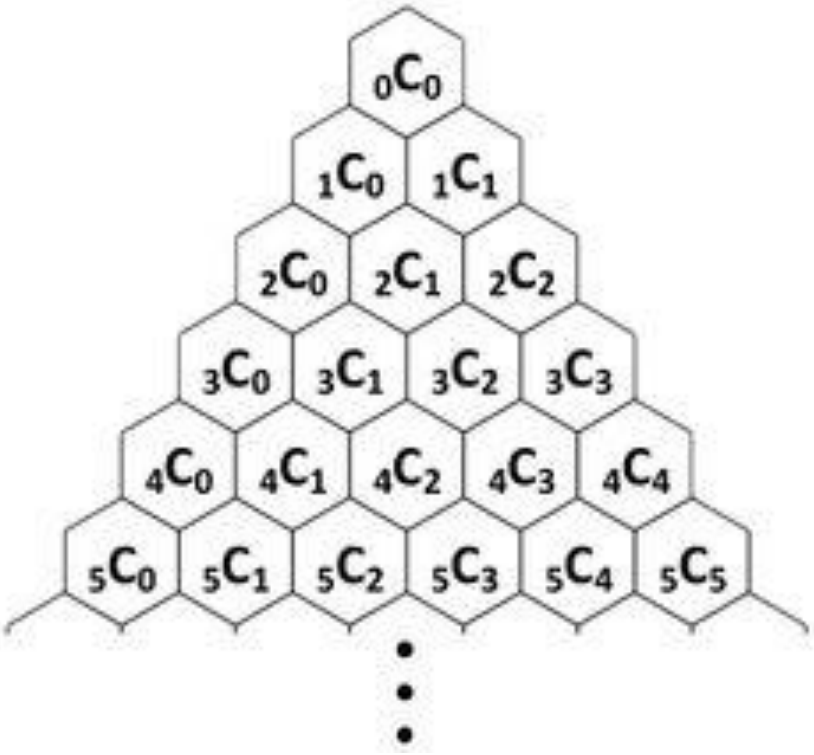
Pascal's Triangle



The image above is pascal’s triangle.

This is nothing but the coefficients of the terms of binomial expansion for different powers.

Pascal’s triangle is a never-ending equilateral triangle of numbers that follow a rule of adding the two numbers above to get the number below. Two of the sides are “all 1's” and because the triangle is infinite, there is no “bottom side.”



Approach for program for pascal’s triangle:

The number of entries in every line is equal to line number. For example, the first line has “1“, the second line has “1 1“, the third line has “1 2 1“,.. and so on. Every entry in a line is value of a Binomial Coefficient. The value of ith entry in line number line is C(line, i). The value can be calculated using following formula.

C(line, i) = line! / ((line-i)! * i!)

Program for pascal's triangle:

```cpp
#include <iostream>
using namespace std;
int binomialCoeff(int n, int k);
void printPascal(int n)
{
    for (int line = 0; line < n; line++) {
        for (int i = 0; i <= line; i++)
            cout << " " << binomialCoeff(line, i);
        cout << "\n";
    }
}
 int binomialCoeff(int n, int k)
{
    int res = 1;
    if (k > n - k)
        k = n - k;
    for (int i = 0; i < k; ++i) {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}
int main()
{
    int n = 7;
    printPascal(n);
    return 0;
}
```

Problems to practice for combinatorics:

1. https://leetcode.com/problems/isomorphic-strings/description/?envType=daily-question&envId=2024-04-02
2. https://leetcode.com/problems/sum-of-all-subset-xor-totals/description/
3. https://leetcode.com/problems/distribute-candies-among-children-i/description/
4. https://leetcode.com/problems/count-sorted-vowel-strings/description/
5. https://codeforces.com/problemset/problem/1743/A
6. https://codeforces.com/problemset/problem/1855/B
7. https://codeforces.com/problemset/problem/478/B
8. https://leetcode.com/problems/find-triangular-sum-of-an-array/description/
9. https://leetcode.com/problems/distribute-candies-among-children-ii/description/
10. https://codeforces.com/problemset/problem/1840/C
11. https://codeforces.com/problemset/problem/1539/A
12. https://codeforces.com/problemset/problem/630/C
13. https://codeforces.com/problemset/problem/1328/B
14. https://codeforces.com/problemset/problem/1931/D
15. https://leetcode.com/problems/unique-paths/description/
16. https://leetcode.com/problems/poor-pigs/description/
17. https://leetcode.com/problems/probability-of-a-two-boxes-having-the-same-number-of-distinct-balls/description/
18. https://leetcode.com/problems/kth-smallest-instructions/description/