n_{Cr} : Combinatorics

- **Binomial Co-efficient: O(n*n)**
- The key idea is $nCr = {}_{n-1}C_{r-1} + {}_{n-1}C_{r}$
- The advantage with this formula is you can modulo ans with any number even though the number isn't prime.(nCr % P)
- Pascal triangle :

1	0	0	0	0	0
2	1	0	0	0	0
3	3	1	0	0	0
4	6	4	1	0	0
5	10	10	5	1	0
6	15	20	15	6	1

- nCr=1 when n==r or r==0;
- nCr=0 when r>n;

Code: loop and recursive solution

```
int C[1001][1001];

void triangel()
{
      C[0][0]=1;

      for(int i=1;i<=1000;i++)
      {
          C[i][0]=1;

          for(int j=1;j<=i;j++)
          {
               C[i][j]=(C[i-1][j-1] + C[i-1][j]);
          }
      }
}
```

```
int nCr(int n, int r)
{
     if(n==r || r==0)return 1;

     if(dp[n][r])return dp[n][r];

     dp[n][r]=nCr(n-1,r-1)+nCr(n-1,r);

     return dp[n][r];
}
```

- Now the problem is you have to count nCr % P , where P is a prime. n,r<=10^6.

$$nC_r = \frac{n!}{(n-r)!\,r!} \;\% \; P$$

$$nC_r = (\, n! * \;(\,(n-r)!\;\; r!\,)^{-1}\,)\%P$$

Now we will have to count n! and the inverse factorial of (n-r)!r! .. we can compute all factorial values in linear time, O(n).

$$(n!)\;\%\;P = (n * (n-1)!\,)\%P$$

```
long long  fact[1000001];

 fact[0]=1;

 for(int i=1; i<=1000000;i++)
   {
       fact[ i ] = ( 1LL * fact[i-1] * i  ) % P;
   }
```

We've generated the values of (n!)%P for any n(0<=n<=1000000). Now it's time to count the inverse factorials. We will learn two approaches. First I will go with the easy one.

$$(\, n!\,)^{-1}\;\%\;P = (\,(n-1)!^{-1} * n^{-1}\;\;)\%P$$

Now we will need the value of $n^{-1}\%P$ to count inverse factorials. The only way I know is use Big Mod theory.

$$n^{-1}\;\%\;P = \;n^{P-2}\;\%\;P$$,where P is Prime

So inverse factorials can be written as ifact[i]= (ifact[i-1] * inv(i))% P;

```
long long  mpower (long long  b, long long  p, long long   mod)
{
      if  (p==0) return 1;
      long long  tmp = mpower(b, p/2, mod);
      tmp= ( tmp * tmp )%mod;

      return (p%2==0)?  tmp : (b* tmp)%mod;
}
long long  inv(long long n, long long  mod)
{
     return mpower(n, mod-2, mod);

}
```

```
long long ifact[1000001];
ifact[0]=1;

for(int i =1;i<=1000000;i++)
{
      ifact[ i ] = ( 1LL *  ifact[ i-1 ] * inv(i , P) )%P;
}
```

The complexity of this generation is O (n log n). Actually it's O (n log P). The inv( ) function takes log P time to generate value.

Now as we have both factorials and inverse factorials. So its time to compute nCr.

```
long long nCr ( long long n, long long r)
{
        if(r>n)return 0;
        long long ans;
        ans=( (1LL * fact[n] * ifact[n-r] )%P * ifact[r] ) %P;
        return ans;

}
```

There is a problem with inverse factorials, it takes O(nlogn) time. We can reduce it to O(n). Look at the previous equation

$$(n!)^{-1} \,\%\, P = ((n-1)!^{-1} * n^{-1} \ )\%P$$

We can generate all $1^{-1} \% P$, $2^{-1} \% P$, $3^{-1} \% P$, … … … , $n^{-1} \% P$ in O(n) time. Let's see little math, using the division theorem P can be written as $P = qk + r$, where 0<= r <k. r =(P % k) and q=(P/k).

$$P = qk + r$$

$$0 \equiv qk + r \ \ (mod\ P)$$

$$k^{-1} = \ -q\, r^{-1}\, (mod\ P)$$

As we know r<k ,so we already have that one.

```
in[0] = 0, in[1] = 1;
for(int i=2; i<=1000000; i++)
{
      in[ i ] = (1LL * (  (P-1)* (P/i)  )%P * in[ P%i ] )%P;
}
ifact2[0] = 1;
for(int i=1; i<=1000000; i++)
{
      ifact2[ i ]= (1LL * ifact2[ i-1 ] * in[ i ] )%P;
}
```

- Now the problem is you have to compute nCr % P, where n,r<=10^18 but P<=10^6.
- We need to solve this problem using Lucas theorem. Because in Lucas theorem problem will be reduced to sub problems. In this theorem the n and r are converted to P base number and then we compute the same digit-location wise binomial coefficients. Lucas theorem is given below :

$$\binom{n}{r} \equiv \prod_{i=0}^{k} \binom{n_i}{r_i} \ (mod\ P)$$

Where n $=(n_k \dots \dots\ n_2\ n_1\ n_0\ )_P$, r $=(r_k \dots \dots\ r_2\ r_1\ r_0\ )_P$

```
long long in[1000001], fact[1000001], ifact[1000001];

void generate(long long MX, long long P)
{
          fact[0]=1;
          for(int i=1;i<=MX;i++) fact[i]=(1LL * fact[i-1] * i)%P;
          in[0]=0,in[1]=1;
          for(int i=2;i<=MX;i++) in[i]= (1LL * ( (P-1)* (P/i) )%P * in[P%i] )%P;
          ifact[0]=1;
          for(int i=1;i<=MX;i++) ifact[i]=(1LL * ifact[i-1] * in[i] )%P;
}

long long small_nCr(long long n, long long r, long long P)
{
         if(r>n)return 0;

         long long ans;
         ans=(1LL * fact[n] * ifact[n-r])%P;
         ans=( ans * ifact[r] )%P;

         return ans;
}

long long nCr(long long n, long long r, long long P)
{
  if(r==0)return 1;

  long long ni=(n%P), ri=(r%P);

  return ( nCr(n/P, r/P, P)* small_nCr(ni, ri, P)) %P;
}

long long Lucas(long long n, long long r, long long P)
{
         generate(P, P);
         return nCr(n, r, P);
}
```

```
int main()
{

    long long prime[]={13,29,67,113,157,223};
    for(int i=0;i<6;i++)
    {
        cout<<( 126%prime[i] )<<'  '<<Lucas(9, 5, prime[i] )<<endl;
    }
}
```

- Now the problem is, if we want to find nCr % P, where **P is not a prime** number. Solution: We can split P into its prime divisors and count binomial coefficients for each divisor and marge them using Chinese remainder theorem.

Chinese Remainder Theorem:

P can be written as, $P = P_1 P_2 \dots \dots P_{n-1} P_n$(prime divisors)

Now we can separately calculate,

$$n_{C_r} = X$$

$$X \equiv\ r_1 \ (mod\ P_1)$$

$$X \equiv\ r_2 \ (mod\ P_2)$$

$$\dots \dots \dots$$

$$X \equiv\ r_n \ (mod\ P_n)$$

Now we can marge the above equations using Chinese remainder theorem. By using this theorem we can find the minimum value of X for which all the equation are true along with the below one.

$$X \equiv\ r\ (mod\ P)$$

X can found by using followed equation,

$$X = \sum_{i=1}^{n} (r_i * pd_i * inv(pd_i\,, P_i))$$

$$pd_i = \frac{P}{P_i} \quad \&\ r = X\%P$$

```
long long  n, prime[20], rim[20];

long long  bigMod(long long  b, long long  p, long long  M)
{
        if(p==0)return 1;

    long long  tmp= bigMod (b, p/2,M);

    tmp = (tmp * tmp)% M;

    return (p%2==0)? tmp : ( b * tmp) % M;

}

long long  INV(long long  num, long long M)
{
        return bigMod(num,M-2,M);
}



long long ChineseRemainder( )
{
     long long  product=1, x=0, pd;
     for(int i=0;i<n;i++)
          product*=prime[i];

     for(int i=0;i<n;i++)
     {

           pd=product/prime[i];

           x+=( rim[i] * pd * INV( pd, prime[i]) );

           x%=product;
     }

return x;

}
```