Coefficient of xk is

* Linear Recurrence:

GF: A(x).

Recurrence: A(i) = C0Ai-1+C1Ai-2+…+Ck-1Ai-k

Let Q(x) = 1-C0x-C2x2-…-Ck-1xk-1. We can show each k -order recurrence can be described by A(x) = P(x) / Q(x) such that

* Q(x) is of degree k, and contains the coefficients of the recurrence. Additionally, the constant term is 1 (or Q(0) = 1).
* P(x) is of degree < k.

The generating functions P(x) / Q(x) and P(x)R(x) / Q(x)R(x) generate the same sequence. If we let R(x) = Q(-x) then all odd terms of the denominator will be vanished.

Another way to look at it:

Let M(x) = xk-C0xk-1-C1xk-2-…-Ck-1

Let mlxl + ml-1xl-1 + ml-2xl-2 + … + m0 be any polynomial divisible by M(x). Then:

*ml*​*Al*​+*ml*−1​*Al*−1​+*ml*−2​*Al*−2​+⋯+*m*0​=0

S(x) = *sk*−1​*xk*−1+*sk*−2​*xk*−2+⋯+*s*1​*x*+*s*0 = ​xN % M(x).

Then AN=Ak-1Sk-1+Ak-2Sk-2+…+A1S1+A0S0

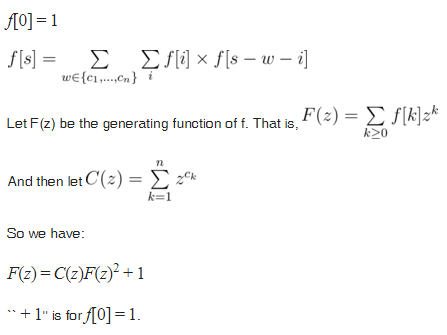
A LR can be shown by the help of many different recurrences.

If A(n) = then,

xN % M(x) = % M(x)

* The generating function for the Catalan numbers is
* Fibonacci Numbers:
* Catalan numbers: 

, [+1 is beacause C[0] = 1]

* 
* **Generating function of a linear recurrence**

// a[k] = c[0] \* a[k - 1] + c[1] \* a[k - 2] + ... for k >= n

mint yo(vector<mint> a, vector<mint> c) {

int n = a.size();

if (!n) return 0;

vector<mint> p(n + 1, 0);

p[0] = 1;

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

p[i + 1] = -c[i];

}

vector<mint> up(n + n, 0);

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

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

up[i + j] += a[i] \* p[j];

}

}

up.resize(n);

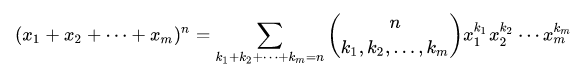
// generating function of the recurrence is up / p

}

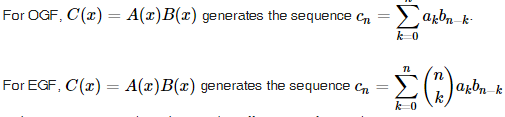
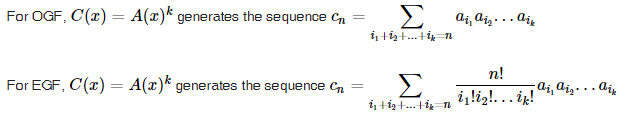
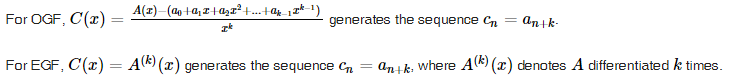
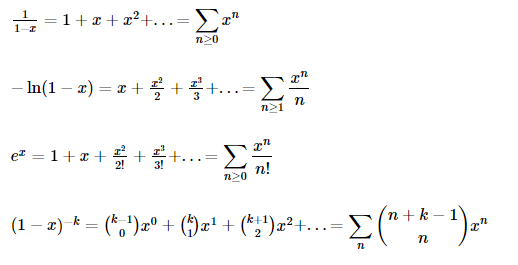
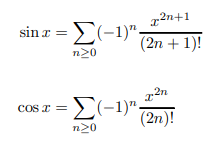
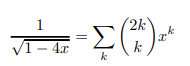
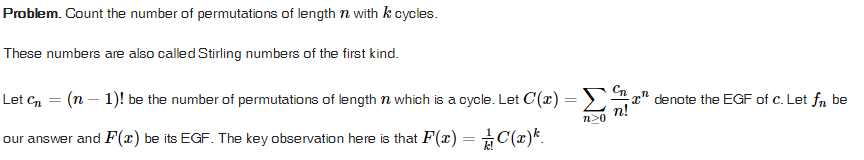
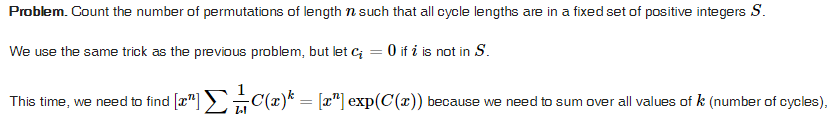
* 
* Find 

This is equivalent to the coefficient of  Thus the solution is the coefficient of xN of the above eqn, which on expanding binomially we get   thus coefficient of xN in the above eqn will be, multiplied by 

* 
* Bell number- number of partitions of a set.

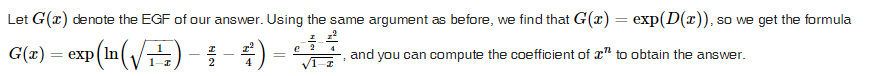


* 
* 
* 
* 
* 
* 
* 
* 
* 
* 
* 



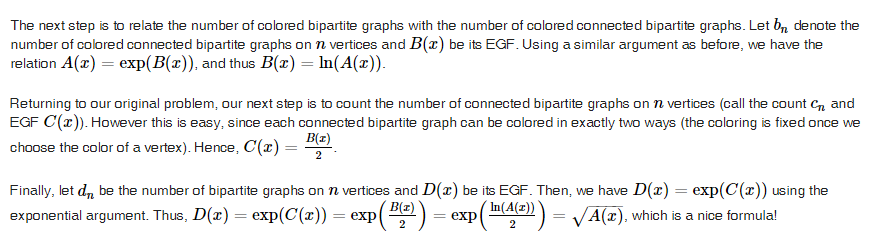
* 





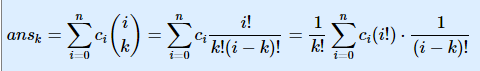
* 

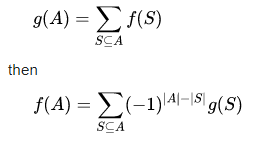




* 
* 





* 
* 



* 
*  = \*xi where 

To be precise,

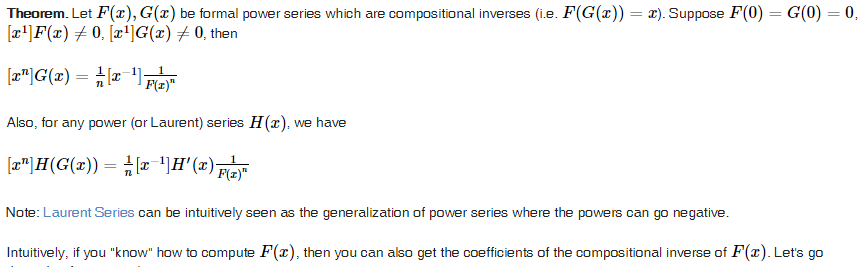
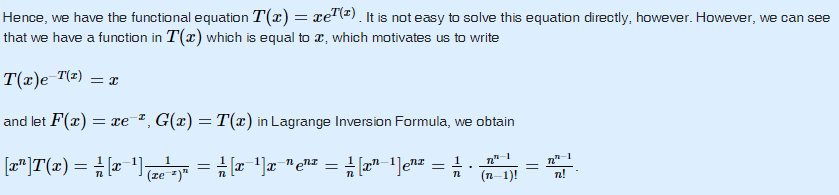
Let dp[i][j]=ways of taking i elements having sum j

if(k<=n) dp[i][j]=dp[i][j-i]+dp[i-1][j-i]

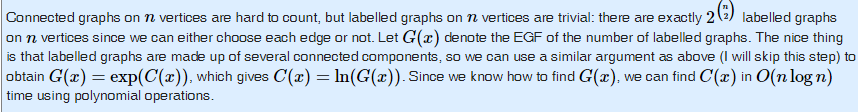
if(k can be >=n) dp[i][j]=dp[i][j-i]+dp[i-1][j-i]-dp[i-1][j-n-1]

Note that here i<=close to sqrt(k)

So we can solve it in O(ksqrt(k))

* OGF of fibonacci sequence is 
* 
* 
* count the number of labelled connected graphs on n vertices

Let C(x) be the EGF of the number of connected labelled graphs.



* it looks like if we want to get a disjoint set of elements, we use exp() and if we want to connect the disjoint set into a single element, we use ln().
* 

From previous problem C(x) = ln(G(X)). Let 



* Coefficient of x^n in is the same as the number of non-negative integer soluions of the eq a + 2b + 4c = n. The other equations are similar