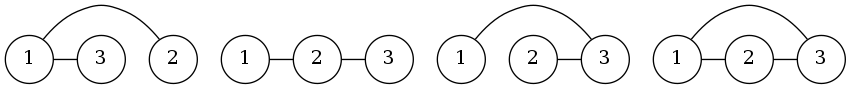
# **[bzoj3456城市规划 多项式取模](https://www.cnblogs.com/Skyminer/p/6393825.html)**

题目要求求出有n(n≤130000)个点的有标号简单连通无向图的个数，答案mod 1004535809 (479×221+1)，是个质数

比如三个点的情况



题目链接：[BZOJ-3456](http://www.lydsy.com/JudgeOnline/problem.php?id=3456" \t "http://blog.miskcoo.com/2015/05/_blank)

首先可以设f(n)表示有n个点的有标号简单连通无向图的个数，g(n)表示有n个点的有标号简单无向图的个数（也就是不要求连通）

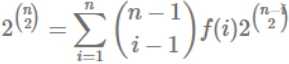
g(n)是很好求的，因为n个点，最多条边，因此



又因为一个有标号简单无向图是由很多连通分量组成的，为了避免重复计数，我们枚举点1所在的连通块大小（其余的点随便连边，因为1号点所在连通块已经确定，其它怎么连都不会重复）



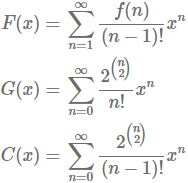
我们把g(n)代入



然后两边同时除以(n−1)!



现在你会发现这是个卷积的形式！定义函数F(x),G(x),C(x)



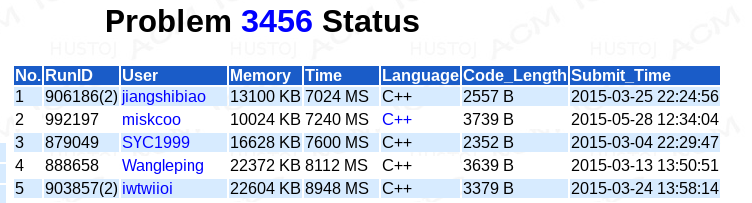
可以得到



将其放在mod xn+1下



这样求出G(x)的逆元然后和C(x)乘起来就可以得到F(x)，也就是答案了，由于模数十分特殊，可以利用FFT优化（多项式求逆看[这里](http://blog.miskcoo.com/2015/05/polynomial-inverse" \t "http://blog.miskcoo.com/2015/05/_blank)）

不知道为什么似乎跑得好快？

#include <cstdio>

#include <algorithm>

using std::swap;

using std::fill;

using std::copy;

using std::reverse;

using std::reverse\_copy;

typedef int value\_t;

typedef long long calc\_t;

const int MaxN = 1 << 18;

const value\_t mod\_base = 479, mod\_exp = 21;

const value\_t mod\_v = (mod\_base << mod\_exp) + 1;

const value\_t primitive\_root = 3;

int epsilon\_num;

value\_t eps[MaxN], inv\_eps[MaxN];

value\_t dec(value\_t x, value\_t v) { x -= v; return x < 0 ? x + mod\_v : x; }

value\_t inc(value\_t x, value\_t v) { x += v; return x >= mod\_v ? x - mod\_v : x; }

value\_t pow(value\_t x, value\_t p)

{

value\_t v = 1;

for(; p; p >>= 1, x = (calc\_t)x \* x % mod\_v)

if(p & 1) v = (calc\_t)x \* v % mod\_v;

return v;

}

void init\_eps(int num)

{

epsilon\_num = num;

value\_t base = pow(primitive\_root, (mod\_v - 1) / num);

value\_t inv\_base = pow(base, mod\_v - 2);

eps[0] = inv\_eps[0] = 1;

for(int i = 1; i != num; ++i)

{

eps[i] = (calc\_t)eps[i - 1] \* base % mod\_v;

inv\_eps[i] = (calc\_t)inv\_eps[i - 1] \* inv\_base % mod\_v;

}

}

void transform(int n, value\_t \*x, value\_t \*w = eps)

{

for(int i = 0, j = 0; i != n; ++i)

{

if(i > j) swap(x[i], x[j]);

for(int l = n >> 1; (j ^= l) < l; l >>= 1);

}

for(int i = 2; i <= n; i <<= 1)

{

int m = i >> 1, t = epsilon\_num / i;

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

{

for(int p = 0, q = 0; p != m; ++p, q += t)

{

value\_t z = (calc\_t)x[j + m + p] \* w[q] % mod\_v;

x[j + m + p] = dec(x[j + p], z);

x[j + p] = inc(x[j + p], z);

}

}

}

}

void inverse\_transform(int n, value\_t \*x)

{

transform(n, x, inv\_eps);

value\_t inv = pow(n, mod\_v - 2);

for(int i = 0; i != n; ++i)

x[i] = (calc\_t)x[i] \* inv % mod\_v;

}

struct poly\_t

{

int deg;

value\_t x[MaxN];

poly\_t() : deg(0) { x[0] = 0; }

};

void polynomial\_inverse(int n, const poly\_t& A, poly\_t& B)

{

if(n == 1)

{

B.deg = 1;

B.x[0] = pow(A.x[0], mod\_v - 2);

return;

}

static value\_t X[MaxN];

polynomial\_inverse((n + 1) >> 1, A, B);

int p = 1;

for(; p < n << 1; p <<= 1);

copy(A.x, A.x + n, X);

fill(X + n, X + p, 0);

transform(p, X);

fill(B.x + B.deg, B.x + p, 0);

transform(p, B.x);

for(int i = 0; i != p; ++i)

B.x[i] = (calc\_t)B.x[i] \* dec(2, (calc\_t)X[i] \* B.x[i] % mod\_v) % mod\_v;

inverse\_transform(p, B.x);

B.deg = n;

}

poly\_t A, B, C;

value\_t inv[MaxN], inv\_fac[MaxN], choose[MaxN];

int main()

{

int n;

std::scanf("%d", &n);

int p = 1;

for(; p < (n + 1) << 1; p <<= 1);

init\_eps(p);

inv[1] = inv\_fac[0] = 1;

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

{

if(i != 1) inv[i] = -mod\_v / i \* (calc\_t)inv[mod\_v % i] % mod\_v;

if(inv[i] < 0) inv[i] += mod\_v;

inv\_fac[i] = (calc\_t)inv\_fac[i - 1] \* inv[i] % mod\_v;

}

choose[0] = choose[1] = 1;

for(int i = 2; i <= n; ++i)//

choose[i] = pow(2, (calc\_t)i \* (i - 1) / 2 % (mod\_v - 1));

A.deg = B.deg = n + 1;

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

A.x[i] = (calc\_t)choose[i] \* inv\_fac[i] % mod\_v;

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

B.x[i] = (calc\_t)choose[i] \* inv\_fac[i - 1] % mod\_v;

polynomial\_inverse(n + 1, A, C);//A-1

fill(C.x + C.deg, C.x + p, 0);

transform(p, C.x);

transform(p, B.x);

for(int i = 0; i <= p; ++i)

C.x[i] = (calc\_t)C.x[i] \* B.x[i] % mod\_v;

inverse\_transform(p, C.x);//

value\_t ans = (calc\_t)C.x[n] \* pow(inv\_fac[n - 1], mod\_v - 2) % mod\_v;

if(ans < 0) ans += mod\_v;

std::printf("%d\n", ans);

return 0;

}