NTT_2D使用说明
接口说明
Poly _ 扩展版 _ 使用说明
【重要数组变量和接口说明】
【不可操作序列修正方法和其他备注】
PAM使用说明
【数组约定】
【函数说明】
【使用方法】
【注意点】

# NTT_2D使用说明

## 接口说明

**1）FT部分：**

void Init(int _K,int p)：两个参数为用户所期望的结果最大次数，和模数

void init_w(int m)：预处理长度为1<<m的w数组和rev数组

void FFT(vector& A,int m,int op)：变换接口，长度为1<<m，op为0表示普通和卷积，op为1表示差卷积

void multiply(const vector& A,const vector& B,vector *C)：

乘法接口，注意第三个参数传入指针

**2）Matrix部分：**

void Set_m(int _m,int x=0)：将每列resize为m，不足元素填充为x

void Set_n(int _n)：将行扩充为n

void Set(int _n,int _m,int x=0)：设置矩阵规模

**void Transpose()：转置接口**

void Reverse()：翻转接口，不光行翻转，列也翻转

void Shift(int x,int y)：移位接口，将(x,y)设置为矩阵左上角元素，整体移位，不足部分填充0

**void FFT(FT &T,int len,int op)：批量行变换，对每行进行正/逆变换**

void print() const：打印接口

void Normalize(int _p)：规范化接口，保证矩阵中每个元素都严格非负且已模

void Random()：产生随机数，填充此矩阵

==：矩阵可以直接比较判等

**3）Calculator部分：**

void Init(int p)：预处理模数

**void Multiply(const Matrix &A,const Matrix &B,Matrix &C,int op=0):**

**乘法接口，op=0为和卷积，op=1为差卷积**

void Multiply_B(const Matrix &A,const Matrix &B,Matrix &C):

暴力和卷积接口

void Multiply_B_sub(const Matrix &A,const Matrix &B,Matrix &C):

暴力差卷积接口

# Poly _ 扩展版 _ 使用说明

## 【重要数组变量和接口说明】

- inv[],fac[],fac_inv[]：逆元，阶乘和阶乘逆
- init_inv(int n)：预处理逆元
- init_fac(int n)：预处理阶乘和阶乘逆元
- struct FT< V >{}：ntt结构体，此处可以换成fft或者其他模板，但是要求适配接口
    - init_len(int _n)：预处理FT内部长度，产生一个严格大于 _n的2的幂给n
    - Init(int _n)：FT预处理，调用之后才可以用FFT()接口
    - void FFT(V A[],int op)：op=0，把A转化为点值；op=1，为其逆
- **void Fill(V a[],V b[],int n,int len)**：标准填充接口，把b的前n个元素赋值给a的前n个元素，且将a中[n,len)清0
- void Add(V a[],int n,V b[],int m,V c[],int t=1)：长度为n(下标[0,n))的多项式a和长度为m的多项式b，计算c[i]=a[i]+t*b[i]；**[注意]**不保证不对齐位置清0
- void Dot_Mul(V a[],V b[],int len,V c[])：长度相等的两个数组，对应元素点乘给c[]
- void Dot_Mul(V a[],int len,V v,V c[])：每个元素乘v
- **void Mul(V a[],int n,V b[],int m,V c[])**：多项式乘法，用户不需要考虑传入数组非法位置的元素
- void Int(V a[],int n,V b[])：多项式积分，只保留结果的前n项
- void Der(V a[],int n,V b[])：求导
- **void Inv(V a[],int n,V b[])**：多项式求逆，也就是倒数；[注意]常数项必须可逆
- **void Log(V a[],int n,V b[])**：多项式对数，[注意]由于结果常数项为0，所以要求多项式常数项必须为1
- **void Exp(V a[],int n,V b[])**：多项式exp，[注意]ntt版本常数项必须为0(否则正确结果不在模意义整数范围内)，其他版本没有要求，double或者复数版本
- **void Sqrt(V a[],int n,V b[])**：多项式开根，[注意]常数项要可以被开根，如果是ntt，注意边界要求a[0]的模意义二次剩余，如果不存在就无解；double版本a[0]不能为负
- **void Power(V a[],int n,ll k,V b[])**：多项式乘方，[注意]由于依赖log和exp，多项式a[]要求常数项为1，如果以一串0开头，左移之后常数为1，则可以使用POW接口
- **V Lagrange(V a[],int n,int k)**：求多项式a[]的拉格朗日逆或者复合逆(也就是f(g(x))=x，已知一个，求另一个）的x^k前系数，要求常数项可逆

- void Div(V a[],int n,V b[],int m,V d[],V r[])：多项式除法，d是商，r是余数
- void Sinh(V a[],int n,V b[])，void Cosh(V a[],int n,V b[])：双曲正余弦函数
- **void Dirichlet_Mul(V a[],int n,V b[],int m,V c[],int L)**：两个序列的狄利克雷卷积，保留前L项
- void Der_k(V a[],int n,int k,V b[])：k阶导
- void Int_k(V a[],int n,int k,V b[])：k重积分
- void Grow(V a[],int n,V b[])：多项式生长操作，a[i] * = i
- void Grow_k(V a[],int n,int k,V b[])：k次生长
- void Shl(V a[],int n,int k,V b[])，void Shr(V a[],int n,int k,V b[])：左右移k位，传入传出均是前n项有效
- void To_egf(V a[],int n,V b[])：普通型生成函数转化为指数型生成函数，a[i] * = fac[i]
- void To_ogf(V a[],int n,V b[])：前面的逆操作
- **void Bin_Mul(V a[],int n,V b[],int m,V c[])**：求两个序列的二项卷积
- void POW(V a[],int n,ll k,V b[],int t=0)：允许前面有连续t个0的且以1开头的序列传入
- void Reverse(V a[],int n,V b[])：位置翻转操作
- void Init_Com_Num_H_B(V a[],int n,ll k)：预处理组合数第k行前n项，k<=1e18
- void Init_Com_Num_L_B(V a[],int n,ll k)：预处理组合数第k列前n项，k<=1e18，[注意]前面多余的0去掉了，也就是从k行k列开始
- void Pre_Sum(V a[],int n,V b[])：求前缀和
- **void Pre_Sum_k(V a[],int n,ll k,V b[])**：k次前缀和，k<=1e18
- void Fly(V a[],int n,ll k,V b[])：起飞操作，a[i] * = k^i
- void Crossify(V a[],int n)：序列交错化，奇数项符号取反
- void Diff(V a[],int n,V b[])：向前差分，b[i] = a[i+1] - a[i]
- **void Diff_k(V a[],int n,int k,V b[])**：k次差分，有效范围每次减少1个
- void Get_all_one(V a[],int n)：获得全1序列
- **void Get_exp_x(V a[],int n)**：获得e^x展开式
- void Get_log_1_add_x(V a[],int n)：获得log(1+x)展开式
- void Init_Bell_Num(V a[],int n)：预处理Bell数
- void Init_Bernoulli_Num(V a[],int n)：预处理Bernoulli数
- **Get_Num_Power_Sum(ll n,int k)**：获得自然数等幂和S(n,k)=1^k^+2^k^+...n^k^ ，n<=1e18
- **void Init_Stiriling_Num_2_H_B(V a[],int n,ll k)**：预处理第二类Stiriling数第k行前n项(0..n-1),k<=1e18
- void Init_Stiriling_Num_2_L(V a[],int n,int k)：预处理第二类Stiriling数第k列前n项(去掉连续0)
- void Init_Stiriling_Num_1_L(V a[],int n,int k)：预处理第一类Stiriling数第k列前n项
- void Mod_p(V a[],int n)：序列取模，转化为可输出的格式

# 【不可操作序列修正方法和其他备注】

- 对序列提取一个常数，比如：常数项为-5，不能开根，整个提取-5，再开根，最后乘上根号-5
- 对多项式提取x ^ t，常数项为0，不能pow，那么g(x)^k^ =x^kt^ * f(x)^k^
- 求逆函数，有时候需要结合二次剩余的模板

- 求第一类Stiriling数的一行是x(x+1)..(x+n-1)=x的n次上升阶乘幂展开，需要分治fft/ntt模板或者启发式合并或者倍增
- 分离一个常数，exp中多项式常数要为0，可以把g(x)写成f(x)+c，f(x)常数项为0，求f(x)的exp最后乘上e^c

# PAM使用说明

## 【数组约定】

1.s[]表示当前已经插入到自动机的串，s[0]=-1，真实的字符从s[1]开始，s[]的活动范围是[0,n]

在多串模式中，中间的间隔符，是从-2开始递减的负数，会完全隔绝串间的回文匹配

2.len[i]表示i这个节点表示的回文子串的长度

3.next[i][c] i这个节点，在字符c方向的转移

4.fail[] 失配指针

5.cnt[i] 表示节点i以最长回文后缀出现的前缀下标的个数，通过count()调用，求出每个节点表示的回文串出现的次数

6.dep[i] i这个节点在parent树中的深度，其实际意义是：以i为终止下标的回文后缀的个数

7.id[i] 表示i这个下标（指的是插入串s[]）,所代表的前缀的最长回文后缀在自动机中的节点编号

8.no[i] i这个节点，在插入串s[]出现的最末下标(此处指的是右端点)，用于获取具体的回文串内容

9.last 当前插入的字符生效后，指向最长回文后缀节点，在当前必然是parent树上的叶子节点

10.n 插入的串，字符数0...n

p 任何时刻都表示自动机中的自动机中的最大节点标号+1，节点标号：0..p-1

p-2 为任意时刻本质不同的回文串的个数

str_cnt 多串模式下表示插入的串的个数，串编号从0开始

d0 当前间隔符 M 字符集大小，0..25，默认小写字母

## 【函数说明】

1.int new_node(int *l) 新建节点，回文长度为*l

2.void Init() 每组数据初始化自动机，O（1）

3.int get_fail(int x) 沿着失配指针，获取最长匹配节点

4.void I(int c) 插入字符c，注意是默认小写字母

5.void Insert(char s[],int _n=0) 用户接口，插入串，多串意义下无须考虑间隔符

6.void count() 树dp，可以计算很多内容，默认计算回文串出现的次数

7.ll Q() 用户接口，主操作，或者称主询问

8.单链表中：查询next[x][y] 等价于 next[x].F(y)；修改next[x][y]=z 等价于 next[x].I(y,z)

## 【使用方法】

1.每组数据都要首先Init()

2.多串模式下，直接Insert(s，长度)

3.求全局回文子串的个数，cnt[]求和即可；也可以不进行count(),直接每个节点cnt[]*dep[]求和

求任意前缀回文子串的个数，相当于动态即时查询，需要动态维护答案；

考虑每个节点每次的贡献，其实就是深度，所以在cnt[x]++的时候，将深度dep[x]加入答案，即可回答

也可以这样考虑，叠加每个新插入字符新增的回文后缀的数目，这个结构上等价于dep[x]

## 【注意点】

1.老生常谈，注意多串插入时，N的大小调整，要比总串长大一点

2.count() 调用时，务必要注意：树dp的数组，是否需要开long long；一般，回文串总数目需要long long