# 板子

## 函数

& and //两个位都为1时，结果才为1

| or //两个位都为0时，结果才为0

^ xor //两个位相同为0，相异为1

~ not //0变1，1变0

is\_sort(first,last,cmp) //判断是否有序

count\_if(all(x),[](typename c){return 条件;}) //根据条件计数

unique(all(x)) //并非删除，而是将重复元素放到末尾，返回去重的尾地址

x.erase(unique(all(x),x.end()) //可真正意义上做到去重

## struct

struct node {

int a,b;

node(int a\_ = 0, int b\_ = 0):a(a\_),b(b\_){} //传参

}

## STL

#### vector

vector<type> name //先进先出

###### 添加

.push\_back(const T& x) //向量尾添加一个x

.insert(iterator it,const T& x) //向量迭代器指向元素前添加一个x

.insert(iterator it,int n,const T& x) /\*向量迭代器指向元素前增加n个相同的x \*/

.emplace() //在指定的位置直接生成一个元素,效率更高

.emplace\_back() //在序列尾部生成一个元素，效率更高

###### 删除

.erase(iterator it) ​//删除向量迭代器指向元素

​.erase(iterator first,iterator last) ​//删除向量[first,last)中元素

​.pop\_back()​ //删除向量最后一个元素

​.clear()​ //清空向量所有元素

###### 遍历

.at(int pos) ​//返回pos位置元素的引用

​.front()​ //返回首元素的引用

​.back()​ //返回尾元素的引用

​.begin()​ //返回向量头指针，指向第一个元素

​.end()​ //返回向量尾指针，指向向量最后一个元素的下一个位置

​.rbegin() ​//反向迭代器，指向最后一个元素

​.rend()​ //反向迭代器，指向第一个元素之前的位置

###### 其他

​.swap(vector&) ​//交换两个同类型向量的数据

​.assign(int n,const T& x) ​//设置向量中前n个元素的值为x

​.assign(const\_iterator first,const\_iterator last)​ /\*向量中[first,last)中元素设置成当前向量元素\*/

.empty()​ //判断向量是否为空，若为空，则向量中无元素

​.size()​ //返回向量中元素的个数

#### map / unordered\_map

map<type,type> name //关键字，储存对象,map会排序,自动按key升序

unordered\_map<type,type> name //关键字，储存对象

###### 查询

.find(key) //找到返回对应的储存对象，没找到返回.end())

.count() //返回指定元素出现的次数

.key\_comp() //返回比较元素key的函数

.lower\_bound() //返回键值为给定元素的第一个位置

.upper\_bound() //返回键值为给定元素的第一个位置

.size() //返回map中元素的个数

.value\_comp() //返回比较元素value的函数

###### 删除

.erase(iterator it) //通过一个条目对象删除

.erase(iterator first，iterator last) //删除一个范围

.erase(const Key&key) //通过关键字删除

.clear() //就相当于.erase(name.begin(),name.end())

#### set / unordered\_set

set<type> name //key = value,set会排序,按key升序

###### 添加

.insert() //向set容器中插入元素

.emplace() /\*在当前set容器中的指定位置直接构造新元素,其效果和insert() 一样，但效率更高\*/

.emplace\_hint() /\*在本质上和emplace()在set容器中构造新元素的方式是一样的，不同之处在于使用者必须为该方法提供一个指示新元素生成位置的迭代器，并作为该方法的第一个参数\*/

###### 删除

.erase() //删除set容器中存储的元素

.clear() //清空set容器中所有的元素，即令set容器的size()为0

###### 查询

.begin() /\*返回指向容器中第一个（注意，是已排好序的第一个）元素的双向迭代器,如果set容器用const限定，则该方法返回的是const类型的双向迭代器\*/

.end() /\*返回指向容器最后一个元素（注意，是已排好序的最后一个）所在位置后一个位置的双向迭代器，通常和begin()结合使用。如果set容器用const限定，则该方法返回的是const类型的双向迭代器\*/

.rbegin() /\*返回指向最后一个（注意，是已排好序的最后一个）元素的反向双向迭代器,如果set容器用const限定，则该方法返回的是const类型的反向双向迭代器\*/

.rend() /\*返回指向第一个（注意，是已排好序的第一个）元素所在位置前一个位置的反向双向迭代器,如果set容器用const限定，则该方法返回的是const类型的反向双向迭代器\*/

.find(value) /\*在set容器中查找值为val的元素，如果成功找到，则返回指向该元素的双向迭代器；反之，则返回和end()方法一样的迭代器。另外，如果set容器用const限定，则该方法返回的是const类型的双向迭代器\*/

###### 其他

.equal\_range(value) /\*该方法返回一个pair对象（包含 2 个双向迭代器），其中pair.first和lower\_bound()方法的返回值等价，pair.second和 upper\_bound()方法的返回值等价。也就是说，该方法将返回一个范围，该范围中包含的值为val的元素(set容器中各个元素是唯一的，因此该范围最多包含一个元素)\*/

.empty() //若容器为空，则返回true；否则false

.size() //返回当前set容器中存有元素的个数

.max\_size() /\*返回set容器所能容纳元素的最大个数，不同的操作系统，其返回值亦不相同\*/

.swap() /\*交换 2 个set容器中存储的所有元素。这意味着，操作的 2 个set 容器的类型必须相同\*/

.count(value) /\*在当前set容器中，查找值为val的元素的个数，并返回。注意，由于set容器中各元素的值是唯一的，因此该函数的返回值最大为1 \*/

#### queue

queue<type> name //先进先出

###### 添加和删除

.push(const T& obj) //在queue的尾部添加一个元素的副本

.pop() //删除queue中的第一个元素

###### 查找

.front() //返回 queue 中第一个元素的引用。如果 queue 是常量，就返回一个常引用；如果 queue 为空，返回值是未定义的

.back() //返回 queue 中最后一个元素的引用。如果 queue 是常量，就返回一个常引用；如果 queue 为空，返回值是未定义的

.size() //返回 queue 中元素的个数

.empty() //如果 queue 中没有元素的话，返回 true

.swap(queue<T> &other\_q) //将当前 queue 中的元素和参数 queue 中的元素交换。它们需要包含相同类型的元素

#### ****priority\_queue****

**priority\_queue<type,函数> name //自动排序,从大到小**

**less<type> //从大到小**

**greater<type> //从小到大**

**//priority\_queue<type,vector<type>,greater<type> >**

**//或自定义函数排序规则**

#### ****strack****

strack<type> name //先进后出

###### 添加和删除

.push(const T& val) //先复制val，再将val副本压入栈顶

.push(T&& obj) //以移动元素的方式将其压入栈顶

.emplace(arg...) /\* arg...可以是一个参数，也可以是多个参数，但它们都只用于构造一个对象，并在栈顶直接生成该对象，作为新的栈顶元素\*/

.pop() //弹出栈顶元素

###### 其他

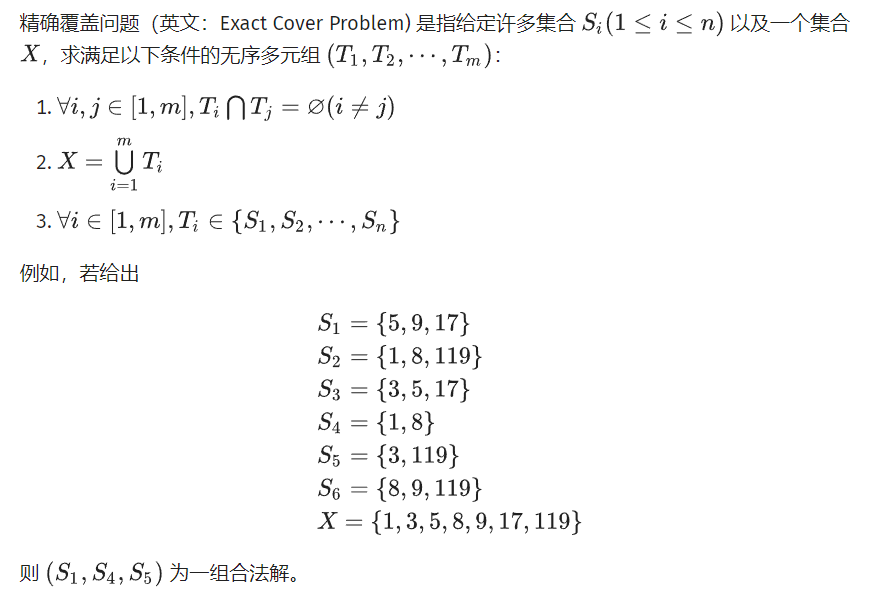
.empty() //当stack栈中没有元素时，该成员函数返回true；反之，返回false

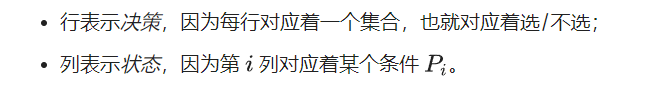
.size() //返回stack栈中存储元素的个数

.top() //返回一个栈顶元素的引用，类型为 T&。如果栈为空，程序会报错

.swap(stack<T> & other\_stack) /\*将两个stack适配器中的元素进行互换，需要注意的是，进行互换的2个stack适配器中存储的元素类型以及底层采用的基础容器类型，都必须相同\*/

## Dancing Links





//离散为01矩阵，复杂度为O(c^n),c接近1,n为1的个数

int n, m, idx, ans, tot;

int stk[N], first[N + M], siz[N + M];

int L[N + M], R[N + M], U[N + M], D[N + M];

int col[N], row[N];

inline void build(int r, int c) {

n = r, m = c;

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

L[i] = i - 1, R[i] = i + 1;

U[i] = D[i] = i;

}

L[0] = c, R[c] = 0, tot = c;

Init<int>(first, 0, N);

Init<int>(siz, 0, N);

}

inline void insert(const int& r, const int& c) {

col[++tot] = c, row[tot] = r, siz[c]++;

D[tot] = D[c], U[D[c]] = tot, U[tot] = c, D[c] = tot;

if (!first[r]) first[r] = L[tot] = R[tot] = tot;

else {

R[tot] = R[first[r]], L[R[first[r]]] = tot;

L[tot] = first[r], R[first[r]] = tot;

}

}

inline void remove(const int& c) {

L[R[c]] = L[c], R[L[c]] = R[c];

for (int i = D[c]; i != c; i = D[i])

for (int j = R[i]; j != i; j = R[j]) {

U[D[j]] = U[j], D[U[j]] = D[j];

siz[col[j]]--;

}

}

inline void recover(const int& c) {

for (int i = U[c]; i != c; i = U[i])

for (int j = L[i]; j != i; j = L[j]) {

U[D[j]] = D[U[j]] = j;

siz[col[j]]++;

}

L[R[c]] = R[L[c]] = c;

}

inline bool dance(int dep) {

if (!R[0]) {

ans = dep;

return 1;

}

int c = R[0];

for (int i = R[0]; i != 0; i = R[i])

if (siz[i] < siz[c]) c = i;

remove(c);

for (int i = D[c]; i != c; i = D[i]) {

stk[dep] = row[i];

for (int j = R[i]; j != i; j = R[j])

remove(col[j]);

if (dance(dep + 1)) return 1;

for (int j = L[i]; j != i; j = L[j])

recover(col[j]);

}

recover(c);

return 0;

}

//int main

n = read<int>(), m = read<int>();

build(n, m);

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

for (int j = 1; j <= m; j++) {

int x = read<int>();

if (x) insert(i, j);

}

dance(1);

if (ans)

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

printf("%d ", stk[i]);

else puts("No Solution!");

## 二分

int a[N];

const ld eps = 1e-6;

inline bool check(ll n) {

}

#### 整数

inline ll judge(ll l, ll r) {

while (l < r) {

int mid = (l + r) >> 1;

if (check(mid)) r = mid;

else l = mid + 1;

}

return l;

}

#### 有精度

inline ld judge(ld l, ld r) {

while (r - l > eps) {

ld mid = (l + r) / 2;

if (check(mid)) r = mid;

else l = mid;

}

return l;

}

#### 无精度

inline ld judge(ld l, ld r) {

for (int i = 1; i <= M; i++) {

double mid = (l + r) / 2;

if (check(mid)) r = mid;

else l = mid;

}

return l;

}

#### 查找

inline int find(ll value) {

ll l = 1, r = 1e9;

while (l <= r) {

ll mid = (l + r) >> 1;

if (a[mid] == value) return mid;

else if (value < a[mid]) r = mid - 1;

else l = mid + 1;

}

return -1;

}

## 快速gcd

inline ull gcd(ull a, ull b) { //调用前需fabs

if (!a) return b;

if (!b) return a;

ull c = ctzll(a | b);

a >>= ctzll(a);

while (b) {

b >>= ctzll(b);

if (a > b) Swap(a, b);

b -= a;

}

return a << c;

}

## 快速幂

inline int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % p;

a = 1ll \* a \* a % p;

b >>= 1;

}

return res;

}

## 除法取模

#### mod为质数

inline int qmod(int a, int b) {

return (a \* qpow(b, mod - 2)) % mod;

}

#### mod不为质数

inline void exgcd(int a, int b, int& x, int& y) {

if (!b) {

x = 1;

y = 0;

return;

}

exgcd(b, a % b, y, x);

y -= (a / b) \* x;

}

//int main

ll a, b, x, y;

exgcd(a, b, x, y);

ll ans = (x % b + b) % b; //逆元

## 排列数与组合数

ll fac[N], ifac[N];

inline int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % p;

a = 1ll \* a \* a % p;

b >>= 1;

}

return res;

}

inline void fact(int n) { //mod需为质数

fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

ifac[n] = qpow(fac[n], mod - 2);

for (int i = n; i >= 1; i--)

ifac[i - 1] = ifac[i] \* i % mod;

}

inline void fact(int n) { //mod可不为质数

fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

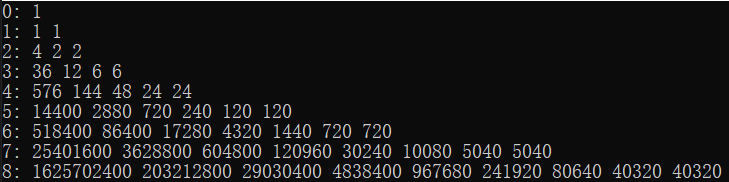
exgcd(fac[n], mod, ifac[n], ifac[0]);

ifac[n] = (ifac[n] % mod + mod) % mod;

for (int i = n - 1; i >= 0; i--)

ifac[i] = ifac[i + 1] \* (i + 1) % mod;

}

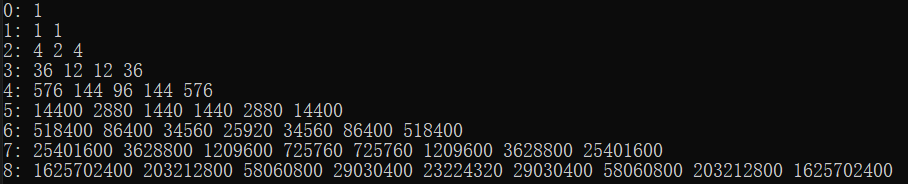


#### inline ll A(int n, int m) { //A(n, m)

if (n < 0 || n < m) return 0;

return fac[n] \* ifac[n - m] % mod;

}



#### inline ll C(int n, int m) { //C(n,m)

if (m < 0 || n < 0 || n < m) return 0;

return fac[n] \* ifac[m] % mod \* ifac[n - m] % mod;

}

//优化？？

ll C[M][M];

#### inline void initC(int n) {

C[0][0] = C[1][0] = C[1][1] = 1;

for (ll i = 2; i <= n; i++) {

C[i][0] = C[i][i] = 1;

for (ll j = 1; j < i; j++)

C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;

}

}

//大组合数对小质数取模（最大p在1e5左右）

#### inline ll Lucas(ll n, ll m, ll p) {

if (m == 0) return 1;

return (C(n % p, m % p, p) \* Lucas(n / p, m / p, p)) % p;

}

//大组合数对小合数取模（最大p在1e5左右）

inline ll calc(ll n, ll x, ll P) {

if (!n) return 1;

ll s = 1;

for (ll i = 1; i <= P; i++)

if (i % x) s = s \* i % P;

s = qpow(s, n / P, P);

for (ll i = n / P \* P + 1; i <= n; i++)

if (i % x) s = i % P \* s % P;

return s \* calc(n / x, x, P) % P;

}

#### inline void exgcd(int a, int b, int& x, int& y) {

if (!b) {

x = 1;

y = 0;

return;

}

exgcd(b, a % b, y, x);

y -= (a / b) \* x;

}

inlinie int inverse(int a, int b) {

ll x, y;

exgcd(a, b, x, y);

ll ans = (x % b + b) % b;

}

inline int multiLucas(int m, int n, int x, int P) {

int cnt = 0;

for (ll i = m; i; i /= x)

cnt += i / x;

for (ll i = n; i; i /= x)

cnt -= i / x;

for (ll i = m - n; i; i /= x)

cnt -= i / x;

return qpow(x, cnt, P) % P \* calc(m, x, P) % P \* inverse(calc(n, x, P), P) % P \* inverse(calc(m - n, x, P), P) % P;

}

inline int CRT(int k, ll\* a, ll\* r) {

ll n = 1, ans = 0;

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

n = n \* r[i];

for (int i = 1; i <= k; i++) {

ll m = n / r[i], b, y;

exgcd(m, r[i], b, y); // b \* m mod r[i] = 1

ans = (ans + a[i] \* m \* b % n) % n;

}

return (ans % n + n) % n;

}

#### inline ll exLucas(ll m, ll n, ll P) {

int cnt = 0;

ll p[20], a[20];

for (ll i = 2; i \* i <= P; i++)

if (P % i == 0) {

p[++cnt] = 1;

while (P % i == 0) {

p[cnt] = p[cnt] \* i;

P /= i;

}

a[cnt] = multiLucas(m, n, i, p[cnt]);

}

if (P > 1) p[++cnt] = P, a[cnt] = multiLucas(m, n, P, P);

return CRT(cnt, a, p);

}

## 分块

int st[M], ed[M], sizes[M], bel[N]; //M需为sqrt(N)

inline void init\_block(int n) // 初始化 {

int sq = sqrt(n);

for (int i = 1; i <= sq; i++) {

st[i] = n / sq \* (i - 1) + 1;

ed[i] = n / sq \* i;

}

ed[sq] = n;

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

for (int j = st[i]; j <= ed[i]; j++)

bel[j] = i;

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

sizes[i] = ed[i] - st[i] + 1;

}

## 数论分块

inlnie ll H(int n) {

ll res = 0;

for (int l = 1, r; l <= n; l = r + 1) {

r = n / (n / l);

res += (r - (l - 1)) \* 1ll \* (n / l); //累加这一块的贡献到结果中。

}

return res;

}

## builtin

#### ctz

int Log2(ui x) {

static const int tb[32] = { 31,0,27,1,28,18,23,2,29,21,19,12,24,9,14,3,30,26,17,22,20,11,8,13,25,16,10,7,15,6,5,4 };

return tb[x \* 263572066 >> 27];

}

int ctz(ui x) { //求x二进制int末尾0的数量

return Log2(lowbit(x));

}

int ctzll(ull x) { //求x二进制ll末尾0的数量

if (!x) return 64;

int r = 63;

x &= ~x + 1;

if (x & 0x00000000FFFFFFFF) r -= 32;

if (x & 0x0000FFFF0000FFFF) r -= 16;

if (x & 0x00FF00FF00FF00FF) r -= 8;

if (x & 0x0F0F0F0F0F0F0F0F) r -= 4;

if (x & 0x3333333333333333) r -= 2;

if (x & 0x5555555555555555) r -= 1;

return r;

}

#### clz

ull int\_block\_positive(ull x) {

ull y = ((x & 0xF0F0F0F0) >> 4) | (x & 0x0F0F0F0F);

y = (((y | 0x80808080) - 0x01010101) >> 7) & 0x01010101;

return y;

}

int int\_block\_first(ull x) {

ull y = x \* 0x01010101ull >> 24;

y = int\_block\_positive(y) \* 0xFF;

return ctzll(y + 1) - 8;

}

int clz(ull x) { //求x二进制ll前导0的数量

if (x == 0) return 64;

int c = 0;

if (x >> 32) x >>= 32;

else c += 32;

ull y = int\_block\_positive(x);

int c1 = int\_block\_first(y);

c += 31 - c1;

x = (x >> c1) & 0xFF;

if (x >> 7) return c - 7;

y = x \* 0x0101010101010101ull | 0x8080808080808080ull;

y = ((y - 0x8040201008040201ull) >> 7) & 0x0101010101010101ull;

if (y >> 32) {

y >>= 32;

c1 = 32;

}

else c1 = 0;

c1 += int\_block\_first(y);

return c - (c1 >> 3);

}

## 二进制翻转

inline int rev(int x) {

int res = 0;

for (int i = 0; i < 30; i++)

res |= ((x >> i) & 1) << (30 - i);

return res;

}

## 根号

inline ld Sqrt(ld n) {

const ld eps = 1E-15; //手动设置

ld x = 1;

while (1) {

ld nx = (x + n / x) / 2;

if (fabsl(x - nx) < eps) break;

x = nx;

}

return x;

}

inline ll isqrt(ll n) {

ll x = 1;

bool decreased = false;

while (1) {

ll nx = (x + n / x) >> 1;

if (x == nx || (nx > x && decreased)) break;

decreased = nx < x;

x = nx;

}

return x;

}

## 欧拉函数值

ll euler\_phi(ll n) { //单个数的欧拉函数值

ll ans = n;

for (int i = 2; i \* i <= n; i++)

if (n % i == 0) {

ans = ans / i \* (i - 1);

while (n % i == 0)

n /= i;

}

if (n > 1) ans = ans / n \* (n - 1);

return ans;

}

## 筛法

#### 素数筛

int f[M];

bool ss[M];

vector<int> primes;

inline void initP(int n) {

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

if (!ss[i]) {

primes.push\_back(i); //选出素数

f[i] = i;

}

for (int j = 0; j < primes.size() && primes[j] \* i <= n; j++) {

ss[primes[j] \* i] = true;

f[primes[j] \* i] = primes[j];

if (i % primes[j] == 0) break;

}

}

}

#### 欧拉筛

int prime[N], phi[N];

bool isPrime[N];

inline void pre() {

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

isPrime[i] = true;

int cnt = 0;

isPrime[1] = false;

phi[1] = 1;

for (int i = 2; i < N; i++) {

if (isPrime[i]) {

prime[++cnt] = i;

phi[i] = i - 1;

}

for (int j = 1; j <= cnt && i \* prime[j] < N; j++) {

isPrime[i \* prime[j]] = false;

if (i % prime[j]) phi[i \* prime[j]] = phi[i] \* phi[prime[j]];

else {

phi[i \* prime[j]] = phi[i] \* prime[j];

break;

}

}

}

}

#### 莫反筛

int mu[N], p[N];

bool v[N];

inline void pre() {

int tot = 0;

Init<bool>(v, false, N);

mu[1] = 1;

for (int i = 2; i <= 1e7; i++) {

if (!v[i]) {

mu[i] = -1;

p[++tot] = i;

}

for (int j = 1; j <= tot && i <= 1e7 / p[j]; j++) {

v[i \* p[j]] = true;

if (i % p[j] == 0) {

mu[i \* p[j]] = 0;

break;

}

mu[i \* p[j]] = -mu[i];

}

}

}

#### 因数筛

int d[N], p[N], num[N];

bool v[N];

//di表示i的因数个数

inline void pre() {

Init<bool>(v, false, N);

int tot = 0;

d[1] = 1;

for (int i = 2; i < N; i++) {

if (!v[i]) {

v[i] = 1;

p[++tot] = i;

d[i] = 2;

num[i] = 1;

}

for (int j = 1; j <= tot && i < N / p[j]; j++) {

v[p[j] \* i] = 1;

if (i % p[j] == 0) {

num[i \* p[j]] = num[i] + 1;

d[i \* p[j]] = d[i] / num[i \* p[j]] \* (num[i \* p[j]] + 1);

break;

}

else {

num[i \* p[j]] = 1;

d[i \* p[j]] = d[i] \* 2;

}

}

}

}

#### 因数和筛

int g[N], f[N], p[N];

bool v[N];

//fi为i的因数和

inline void pre() {

Init<bool>(v, false, N);

int tot = 0;

g[1] = f[1] = 1;

for (int i = 2; i < N; i++) {

if (!v[i]) {

v[i] = 1;

p[++tot] = i;

g[i] = i + 1;

f[i] = i + 1;

}

for (int j = 1; j <= tot && i < N / p[j]; j++) {

v[p[j] \* i] = 1;

if (i % p[j] == 0) {

g[i \* p[j]] = g[i] \* p[j] + 1;

f[i \* p[j]] = f[i] / g[i] \* g[i \* p[j]];

break;

}

else {

f[i \* p[j]] = f[i] \* f[p[j]];

g[i \* p[j]] = 1 + p[j];

}

}

}

}

## Floyd判环倍增优化

inlne ll Pollard\_Rho(ll x) {

ll s = 0, t = 0;

ll c = rand() % (x - 1) + 1;

int step = 0, goal = 1;

ll val = 1;

for (goal = 1;; goal <<= 1, s = t, val = 1) {

for (step = 1; step <= goal; step++) {

t = (t \* t + c) % x;

val = val \* abs(t - s) % x;

if ((step % 127) == 0) {

ll d = gcd(val, x);

if (d > 1) return d;

}

}

ll d = gcd(val, x);

if (d > 1) return d;

}

}

## 类欧几里得算法

int inv2 = qpow(2, mod - 2);

int inv6 = qpow(6, mod - 2);

struct node {

ll f, g, h;

};

inline int qpow(int a, int b, int p = mod) {

ll res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % p;

a = 1ll \* a \* a % p;

b >>= 1;

}

return res;

}

inline node fgh(ll a, ll b, ll c, ll n) {

node ans, tmp;

if (a == 0) {

ans.f = (n + 1) \* (b / c) % mod;

ans.g = (b / c) \* n % mod \* (n + 1) % mod \* inv2 % mod;

ans.h = (n + 1) \* (b / c) % mod \* (b / c) % mod;

return ans;

}

if (a >= c || b >= c) {

tmp = fgh(a % c, b % c, c, n);

ans.f = tmp.f;

ans.f = (ans.f + (a / c) \* n % mod \* (n + 1) % mod \* inv2 % mod) % mod;

ans.f = (ans.f + (b / c) \* (n + 1) % mod) % mod;

ans.g = tmp.g;

ans.g = (ans.g + (a / c) \* n % mod \* (n + 1) % mod \* (2 \* n + 1) % mod \* inv6 % mod) % mod;

ans.g = (ans.g + (b / c) \* n % mod \* (n + 1) % mod \* inv2 % mod) % mod;

ans.h = tmp.h;

ans.h = (ans.h + (a / c) \* (a / c) % mod \* n % mod \* (n + 1) % mod \* (2 \* n + 1) % mod \* inv6 % mod) % mod;

ans.h = (ans.h + (b / c) \* (b / c) % mod \* (n + 1) % mod) % mod;

ans.h = (ans.h + (a / c) \* (b / c) % mod \* n % mod \* (n + 1) % mod) % mod;

ans.h = (ans.h + 2 \* (a / c) % mod \* tmp.g % mod) % mod;

ans.h = (ans.h + 2 \* (b / c) % mod \* tmp.f % mod) % mod;

return ans;

}

ll m = (a \* n + b) / c;

tmp = fgh(c, c - b - 1, a, m - 1);

ans.f = (n \* (m % mod) % mod - tmp.f) % mod;

ans.g = (n \* (n + 1) % mod \* (m % mod) % mod - tmp.f - tmp.h) % mod \* inv2 % mod;

ans.h = n \* (m % mod) % mod \* ((m + 1) % mod) % mod;

ans.h = (ans.h - 2 \* tmp.g - 2 \* tmp.f - ans.f) % mod;

return ans;

}

## 中国剩余定理

inline void exgcd(int a, int b, int& x, int& y) {

if (!b) {

x = 1;

y = 0;

return;

}

exgcd(b, a % b, y, x);

y -= (a / b) \* x;

}

inline int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % p;

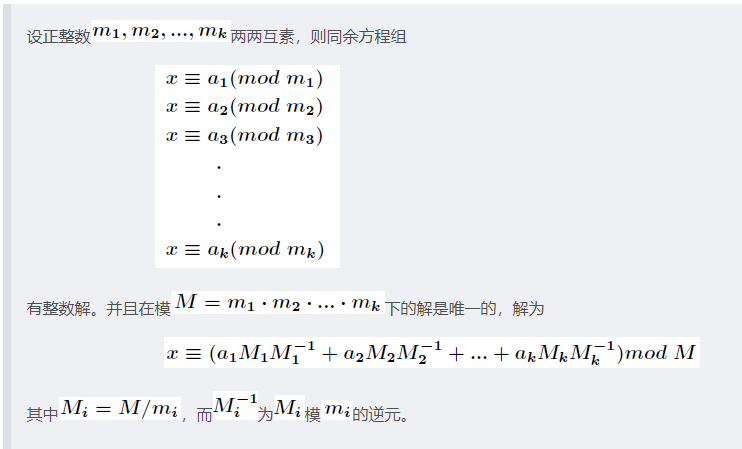
a = 1ll \* a \* a % p;

b >>= 1;

}

return res;

}



//a为同余数，r为mod

inline int Crt(int k, ll\* a, ll\* r) {

ll n = 1, ans = 0;

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

n = n \* r[i];

for (int i = 1; i <= k; i++) {

ll m = n / r[i], b, y;

exgcd(m, r[i], b, y); // b \* m mod r[i] = 1

ans = (ans + a[i] \* m \* b % n) % n;

}

return (ans % n + n) % n;

}

//模数两两不一定互质

inline int excrt(int n, ll\* a, ll\* r) {

ll M = r[1], ans = a[1];

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

ll x = M, y = r[i], z = (a[i] - ans % y + y) % y;

ll gcd, z;

exgcd(x, y, gcd, z);

ll bg = y / gcd;

x = qpow(x, y / gcd, bg);

ans = ans + x \* M;

M \*= bg;

ans = (ans % M + M) % M;

}

return ans;

}

ll x[N], a[N], p[N], r[N][N]; //p是小质数，r为pi在pj意义下的逆元

//用比较小的质数表示大整数

inline void Garner(int n) {

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

x[i] = a[i];

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

x[i] = r[j][i] \* (x[i] - x[j]);

x[i] = x[i] % p[i];

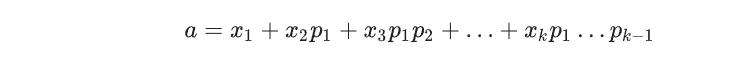
if (x[i] < 0) x[i] += p[i];

}

}

}

//ans



## FFT

const double pi = acos(-1);

struct Complex {

double x, y;

inline Complex(double \_x = 0, double \_y = 0) :x(\_x), y(\_y) {}

inline Complex operator - (const Complex& a)const {

return Complex(x - a.x, y - a.y);

}

inline Complex operator + (const Complex& a)const {

return Complex(x + a.x, y + a.y);

}

inline void operator += (const Complex& a) {

x += a.x;

y += a.y;

}

inline Complex operator \* (const double& a)const {

return Complex(x \* a, y \* a);

}

inline Complex operator \* (const Complex& a)const {

return Complex(x \* a.x - y \* a.y, x \* a.y + y \* a.x);

}

inline Complex operator / (const double& a)const {

return Complex(x / a, y / a);

}

inline Complex operator / (const Complex& a)const {

return Complex(x \* a.x + y \* a.y, a.x \* y - a.y \* x) / (a.x \* a.x + a.y \* a.y);

}

inline Complex conj()const {

return Complex(x, -y);

}

};

using fft = vector<Complex>;

using poly = vector<int>;

namespace FFT {

inline fft Omega(int L) {

fft w(L);

w[1] = 1;

for (int i = 2; i < L; i <<= 1) {

auto w0 = w.begin() + i / 2, w1 = w.begin() + i;

Complex wn(cos(pi / i), sin(pi / i));

for (int j = 0; j < i; j += 2) {

w1[j] = w0[j >> 1];

w1[j + 1] = w1[j] \* wn;

}

}

return w;

}

fft W = Omega(1 << 18);

inline fft DFT(fft a, int n) {

for (int k = n >> 1; k; k >>= 1)

for (int i = 0; i < n; i += k << 1)

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

Complex x = a[i + j];

Complex y = a[i + j + k];

a[i + j + k] = (x - y) \* W[k + j];

a[i + j] = x + y;

}

return a;

}

inline fft IDFT(fft a, int n) {

for (int k = 1; k < n; k <<= 1)

for (int i = 0; i < n; i += k << 1)

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

Complex x = a[i + j];

Complex y = a[i + j + k] \* W[k + j];

a[i + j + k] = x - y;

a[i + j] = x + y;

}

const double Inv = 1.0 / n;

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

a[i].x \*= Inv;

a[i].y \*= Inv;

}

reverse(a.begin() + 1, a.begin() + n);

return a;

}

}

using namespace FFT;

namespace Poly {

int norm(int n) {

return 1 << ((int)log2(n - 1) + 1);

}

poly operator \* (poly& a, poly& b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

int lim = norm(len);

fft c(lim);

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

c[i].x = a[i];

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

c[i].y = b[i];

c = DFT(c, n);

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

c[i] = c[i] \* c[i];

c = IDFT(c, n);

a.resize(n);

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

a[i] = int(c[i].y \* 0.5 + 0.5);

return a;

}

}

using namespace Poly;

## NTT及生成函数

using poly = vector<int>;

namespace Poly {

int inv[N], fac[N << 1], ifac[N << 1];

poly A[N << 2], B[N << 2], C[N << 2];

constexpr int P(998244353), G(3);

inline int INIT(const int& n) {

return 1 << (32 - clz(n - 1));

}

inline int sum(const int& x, const int& y) {

return x + y >= P ? x + y - P : x + y;

}

inline int sub(const int& x, const int& y) {

return x < y ? x - y + P : x - y;

}

inline int mul(const int& x, const int& y, const int& p = P) {

return 1ll \* x \* y % p;

}

inline int qpow(int a, int b = P - 2, const int& p = P) {

int res = 1;

while (b) {

if (b & 1) res = mul(res, a, p);

a = mul(a, a, p);

b >>= 1;

}

return res;

}

inline poly getW(const int& L) {

int wn = qpow(G, P / L);

poly w(L);

w[L / 2] = 1;

for (int i = (L >> 1) + 1; i < L; i++)

w[i] = mul(wn, w[i - 1]);

for (int i = (L >> 1) - 1; i > 0; i--)

w[i] = w[i << 1];

return w;

}

constexpr int wa(911660635), inv2(499122177), wa2(43291859);

//int wa = qpow(G, P - 1 >> 2), inv2 = qpow(2, P - 2, P), wa2 = wa \* inv2 % mod;

poly w = getW(1 << 18);

inline poly dft(poly a, const int& n) {

for (int k = n >> 1; k; k >>= 1)

for (int i = 0; i < n; i += k << 1)

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

int y = a[i + j + k];

a[i + j + k] = mul(sub(a[i + j], y), w[j + k]);

a[i + j] = sum(a[i + j], y);

}

return a;

}

inline poly idft(poly a, const int& n) {

for (int k = 1; k < n; k <<= 1)

for (int i = 0; i < n; i += k << 1)

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

int x = a[i + j], y = mul(a[i + j + k], w[j + k]);

a[i + j] = sum(x, y);

a[i + j + k] = sub(x, y);

}

for (int i = 0, inv = P - (P - 1) / n; i < n; i++)

a[i] = mul(a[i], inv);

reverse(a.begin() + 1, a.begin() + n);

return a;

}

inline poly operator \* (poly a, const int& b) {

int n = a.size();

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

a[i] = mul(a[i], b);

return a;

}

inline poly operator \* (poly a, poly b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

if (n <= 8 || m <= 8) {

poly c(len);

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

for (int j = 0; j < m; j++)

c[i + j] = sum(c[i + j], mul(a[i], b[j]));

return c;

}

//int lim = 1 << (int)log2(len - 1) + 1;

int lim = INIT(len);

a.resize(lim);

b.resize(lim);

a = dft(a, lim);

b = dft(b, lim);

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

a[i] = mul(a[i], b[i]);

a = idft(a, lim);

a.resize(len);

return a;

}

inline poly operator + (poly a, poly b) {

int n = b.size();

a.resize(max((int)a.size(), n));

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

a[i] = sum(a[i], b[i]);

return a;

}

inline poly operator - (poly a, poly b) {

int n = b.size();

a.resize(max((int)a.size(), n));

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

a[i] = sum(a[i], P - b[i]);

return a;

}

inline poly shift(poly a, const int& x) {

if (x < 0) return -x > a.size() ? poly() : poly(a.begin() - x, a.end());

a.insert(a.begin(), x, 0);

return a;

}

inline poly polyInv(const poly& a, const int& n) {

if (n == 1) {

poly b(1);

b[0] = qpow(a[0], P - 2);

return b;

}

poly b = polyInv(a, (n + 1) / 2);

poly c = poly(a.begin(), a.begin() + min(n, int(a.size())));

//int len = 1 << (int)log2(2 \* n - 1) + 1;

int len = INIT(n << 1);

b.resize(len);

c.resize(len);

b = dft(b, len);

c = dft(c, len);

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

b[i] = mul(sub(2, mul(b[i], c[i])), b[i]);

b = idft(b, len);

b.resize(n);

return b;

}

inline poly polySqrt(const poly& a, const int& n) {

poly b{ a[0] }, c;

for (int l = 4; (l >> 2) < n; l <<= 1) {

int r = l >> 1;

b.resize(r);

c = poly(a.begin(), a.begin() + min(r, (int)a.size())) \* polyInv(b, r);

for (int j = 0; j < r; j++)

b[j] = mul(sum(c[j], b[j]), inv2);

}

b.resize(n);

return b;

}

inline poly Derivation(poly a) {

int n = a.size();

if (!n) return a;

for (int i = 0; i + 1 < n; i++)

a[i] = mul(a[i + 1], i + 1);

a.pop\_back();

return a;

}

inline poly Integral(poly a) {

int n = a.size();

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

a[i] = mul(a[i], inv[i + 1]);

a.insert(a.begin(), 0);

return a;

}

inline poly polyLn(poly a, const int& n) {

a = Integral(Derivation(a) \* polyInv(a, n));

a.resize(n);

return a;

}

inline poly polyExp(poly a, const int& n) {

if (n == 1) {

poly b;

b.push\_back(1);

return b;

}

poly b = polyExp(a, (n + 1) / 2);

poly c = b;

b = polyLn(b, n);

int m = b.size();

if (a.size() < m) a.resize(m);

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

b[i] = sum(sub((i == 0), b[i]), a[i]);

b = b \* c;

b.resize(n);

return b;

}

inline poly polyPow(poly a, const int& n, const int& b) {

a = polyExp(polyLn(a, n) \* b, n);

a.resize(n);

return a;

}

inline poly operator / (poly a, poly b) {

int len = 1, deg = a.size() - b.size() + 1;

reverse(all(a));

reverse(all(b));

while (len <= deg)

len <<= 1;

b = polyInv(b, len);

b.resize(deg);

a = a \* b;

a.resize(deg);

reverse(all(a));

return a;

}

inline poly operator % (const poly& a, const poly& b) {

poly c = a - (a / b) \* b;

c.resize(b.size() - 1);

return c;

}

inline poly operator ^ (const poly& a, poly b) {

int n = a.size(), m = b.size();

reverse(all(b));

b = b \* a;

b.resize(n + m);

poly ans(n);

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

ans[i] = b[i + m - 1];

return ans;

}

inline poly operator << (const poly& a, const int& n) {

int m = a.size();

int len = n + m;

poly ans(len);

for (int i = len - 1, j = m - 1; j >= 0; i--, j--)

ans[i] = a[j];

return ans;

}

inline poly polySin(poly a, const int& n) {

//a.resize(n);

a = (polyExp(a \* wa, n) - polyExp(a \* (mod - wa), n)) \* wa2;

return a;

}

inline poly polyCos(poly a, const int& n) {

//a.resize(n);

a = (polyExp(a \* wa, n) + polyExp(a \* (P - wa), n)) \* inv2;

return a;

}

inline poly polytan(const poly& a, const int& n) {

poly c = polySin(a, n) / polyCos(a, n);

return c;

}

inline poly polyAsin(const poly& a, const int& n) {

poly b, c, d;

b = Derivation(a);

c = a \* a;

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

c[i] = sub(mod, c[i]);

c[0] = sum(c[0], 1);

d = polySqrt(c, n);

d = polyInv(d, n);

d = b \* d;

d = Integral(d);

return d;

}

inline poly polyAcos(const poly& a, const int& n) {

poly b = polyAsin(a, n);

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

b[i] = sub(mod, b[i]);

return b;

}

inline poly polyAtan(const poly& a, const int& n) {

poly b, c, d;

b = Derivation(a);

c = a \* a;

c[0] = sum(c[0], 1);

d = polyInv(c, n);

d = b \* d;

d = Integral(d);

return d;

}

inline poly MultiPoint(poly f, poly g) { //多点取值

int n = max(f.size(), g.size()), m = g.size();

f.resize(n);

g.resize(n);

function<void(int, int, int)> build = [&](int k, int l, int r) {

if (l == r) {

A[k] = poly{ 1,(P - g[l]) % P };

return;

}

int mid = (l + r) >> 1;

build(k << 1, l, mid);

build(k << 1 | 1, mid + 1, r);

A[k] = A[k << 1] \* A[k << 1 | 1];

};

build(1, 0, n - 1);

poly ans(n);

function<void(int, int, int, poly)> calc = [&](int k, int l, int r, poly F) {

F.resize(r - l + 1);

if (l == r) {

ans[l] = F[0];

return;

}

int mid = (l + r) >> 1;

calc(k << 1, l, mid, F ^ A[k << 1 | 1]);

calc(k << 1 | 1, mid + 1, r, F ^ A[k << 1]);

};

calc(1, 0, n - 1, f ^ polyInv(A[1], A[1].size()));

ans.resize(m);

return ans;

}

inline poly fdt(poly a, const int& n) {

poly b(n);

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

b[i] = ifac[i];

a = a \* b;

a.resize(n);

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

a[i] = mul(a[i], fac[i]);

return a;

}

inline poly ifdt(poly a, const int& n) {

poly b(n);

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

a[i] = mul(a[i], ifac[i]);

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

if (i & 1) b[i] = mod - ifac[i];

else b[i] = ifac[i];

a = a \* b;

a.resize(n);

return a;

}

inline poly FDT(poly a, poly b) {

int len = a.size() + b.size() - 1;

a.resize(len);

b.resize(len);

a = fdt(a, len);

b = fdt(b, len);

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

a[i] = mul(a[i], b[i]);

a = ifdt(a, len);

a.resize(len);

return a;

}

inline poly toFDT(const poly& a, const int& n) {

poly b(n);

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

b[i] = i;

poly c = MultiPoint(a, b);

c = ifdt(c, n);

return c;

}

inline void FDTto(const poly& a, const int& p, const int& l, const int& r) { //结果为C[1]

if (l == r) {

B[p] = poly({ mod - l , 1 });

C[p] = poly({ a[l] });

return;

}

int mid = (l + r) >> 1;

FDTto(a, p << 1 | 1, mid + 1, r);

FDTto(a, p << 1, l, mid);

B[p] = B[p << 1] \* B[p << 1 | 1];

C[p] = C[p << 1] + C[p << 1 | 1] \* B[p << 1];

}

inline void initInv(const int& n) { //ex时预处理

inv[0] = inv[1] = 1;

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

inv[i] = mul(inv[mod % i], mod - mod / i);

}

inline void initF(const int& n) {

fac[0] = 1;

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

fac[i] = mul(fac[i - 1], i);

ifac[n] = qpow(fac[n]);

for (int i = n; i >= 1; i--)

ifac[i - 1] = mul(ifac[i], i);

}

}

using namespace Poly;

## MTT

#### 三模数NTT(精度高但慢)

int p;

const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, G = 3;

const ll mod12 = 1002772198720536577ll;

const int inv1 = 669690699, inv2 = 354521948;

struct Int {

int A, B, C;

inline Int() :A(), B(), C() {}

inline Int(int \_num) : A(\_num), B(\_num), C(\_num) {}

inline Int(int \_A, int \_B, int \_C) : A(\_A), B(\_B), C(\_C) {}

static inline Int reduce(const Int& x) {

return Int(x.A + (x.A >> 31 & mod1), x.B + (x.B >> 31 & mod2), x.C + (x.C >> 31 & mod3));

}

inline friend Int operator + (const Int& lhs, const Int& rhs) {

return reduce(Int(lhs.A + rhs.A - mod1, lhs.B + rhs.B - mod2, lhs.C + rhs.C - mod3));

}

inline friend Int operator - (const Int& lhs, const Int& rhs) {

return reduce(Int(lhs.A - rhs.A, lhs.B - rhs.B, lhs.C - rhs.C));

}

inline friend Int operator \* (const Int& lhs, const Int& rhs) {

return Int(1ll \* lhs.A \* rhs.A % mod1, 1ll \* lhs.B \* rhs.B % mod2, 1ll \* lhs.C \* rhs.C % mod3);

}

inline int val() {

ll x = 1ll \* (B - A + mod2) % mod2 \* inv1 % mod2 \* mod1 + A;

return (1ll \* (C - x % mod3 + mod3) % mod3 \* inv2 % mod3 \* (mod12 % p) % p + x) % p;

}

};

using ntt = vector<Int>;

namespace MTT {

int lim;

int inv[N], r[N]; //建议开十倍

Int invN, gk[N];

inline int qpow(int a, int b = p - 2, int P = p) {

int res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % P;

a = 1ll \* a \* a % P;

b >>= 1;

}

return res;

}

inline void nttInit(int n) {

int l = 0;

for (lim = 1; lim <= n;) {

lim <<= 1;

l++;

}

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

r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

Int gn(qpow(3, (mod1 - 1) / lim, mod1),

qpow(3, (mod2 - 1) / lim, mod2),

qpow(3, (mod3 - 1) / lim, mod3));

gk[0] = Int(1);

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

gk[i] = gk[i - 1] \* gn;

invN = Int(qpow(lim, mod1 - 2, mod1), qpow(lim, mod2 - 2, mod2), qpow(lim, mod3 - 2, mod3));

}

inline ntt Ntt(ntt a, int opt) {

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

if (r[i] < i) swap(a[i], a[r[i]]);

for (int mid = 1; mid < lim; mid <<= 1) {

int t = lim / (mid << 1);

for (int i = 0; i < lim; i += (mid << 1))

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

Int tmp = a[i + j + mid] \* gk[t \* j];

a[i + j + mid] = (a[i + j] - tmp);

a[i + j] = (a[i + j] + tmp);

}

}

if (opt == -1) {

reverse(a.begin() + 1, a.begin() + lim);

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

a[i] = invN \* a[i];

}

return a;

}

inline ntt operator \* (ntt a, ntt b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

nttInit(len);

a.resize(lim);

b.resize(lim);

a = Ntt(a, 1);

b = Ntt(b, 1);

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

a[i] = a[i] \* b[i];

a = Ntt(a, -1);

a.resize(len);

return a;

}

}

using namespace MTT;

#### 拆系数FFT优化(精度低但快,NTT优化完好像差不多？)

namespace MTT {

inline poly mul(const poly& a, const poly& b, const int& P) {

int n = a.size(), m = b.size();

int len = n + m - 1, l = 1 << ((int)log2(len - 1) + 1);

fft A(l), B(l), c0(l), c1(l);

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

A[i] = Complex(a[i] & 0x7fff, a[i] >> 15);

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

B[i] = Complex(b[i] & 0x7fff, b[i] >> 15);

A = DFT(A, l);

B = DFT(B, l);

for (int k = 1, i = 0; k < l; k <<= 1)

for (; i < k \* 2; i++) {

int j = i ^ k - 1;

c0[i] = Complex(A[i].x + A[j].x, A[i].y - A[j].y) \* B[i] \* 0.5;

c1[i] = Complex(A[i].y + A[j].y, -A[i].x + A[j].x) \* B[i] \* 0.5;

}

c0 = IDFT(c0, l);

c1 = IDFT(c1, l);

poly res(len);

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

ll c00 = c0[i].x + 0.5, c01 = c0[i].y + 0.5;

ll c10 = c1[i].x + 0.5, c11 = c1[i].y + 0.5;

res[i] = (c00 + ((c01 + c10) % P << 15) + (c11 % P << 30)) % P;

}

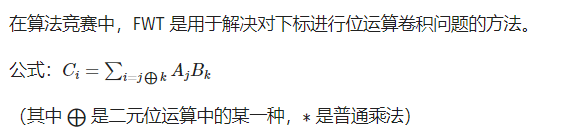
return res;

}

}

using namespace MTT;

## FWT



using namespace Poly;

namespace FWT {

inline poly OR(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

for (int j = 0; j < m; j++)

a[i + j + m] = sum(a[i + j + m], a[i + j]);

return a;

}

inline poly IOR(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

for (int j = 0; j < m; j++)

a[i + j + m] = sub(a[i + j + m], a[i + j]);

return a;

}

inline poly operator | (poly a, poly b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

//int lim = 1 << (int)log2(len - 1) + 1;

int lim = INIT(len);

a.resize(lim << 1);

b.resize(lim << 1);

a = OR(a, lim);

b = OR(b, lim);

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

a[i] = mul(a[i], b[i]);

a = IOR(a, lim);

a.resize(len);

return a;

}

inline poly AND(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

for (int j = 0; j < m; j++)

a[i + j] = sum(a[i + j], a[i + j + m]);

return a;

}

inline poly IAND(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

for (int j = 0; j < m; j++)

a[i + j] = sub(a[i + j], a[i + j + m]);

return a;

}

inline poly operator & (poly a, poly b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

//int lim = 1 << (int)log2(len - 1) + 1;

int lim = INIT(len);

a.resize(lim << 1);

b.resize(lim << 1);

a = AND(a, lim);

b = AND(b, lim);

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

a[i] = mul(a[i], b[i]);

a = IAND(a, lim);

a.resize(len);

return a;

}

inline poly XOR(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

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

int x = a[i + j];

int y = a[i + j + m];

a[i + j] = sum(x, y);

a[i + j + m] = sub(x, y);

}

return a;

}

inline poly IXOR(poly a, const int& n) {

for (int len = 2, m = 1; len <= n; len <<= 1, m <<= 1)

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

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

int x = mul(inv2, a[i + j]);

int y = mul(inv2, a[i + j + m]);

a[i + j] = sum(x, y);

a[i + j + m] = sub(x, y);

}

return a;

}

inline poly Xor(poly a, poly b) {

int n = a.size(), m = b.size();

int len = n + m - 1;

//int lim = 1 << (int)log2(len - 1) + 1;

int lim = INIT(len);

a.resize(lim << 1);

b.resize(lim << 1);

a = XOR(a, lim);

b = XOR(b, lim);

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

a[i] = mul(a[i], b[i]);

a = IXOR(a, lim);

a.resize(len);

return a;

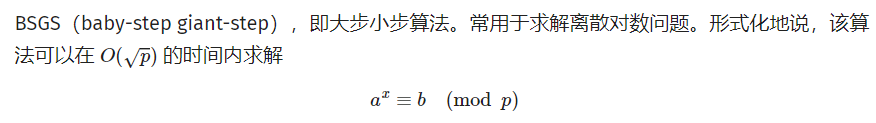
}

}

using namespace FWT;

c(i,j)=(-1)^cnt[i&j] //异或卷积的变换系数,cnt[i]为i中1的个数

## BSGS



inline void exgcd(int a, int b, int& x, int& y) {

if (!b) {

x = 1;

y = 0;

}

else {

exgcd(b, a % b, x, y);

ll t = x;

x = y;

y = t - a / b \* y;

}

}

inline int inv(int a, int b) { //逆元

ll x, y;

exgcd(a, b, x, y);

return (x % b + b) % b;

}

inline int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = 1ll \* res \* a % p;

a = 1ll \* a \* a % p;

b >>= 1;

}

return res;

}

inline void bsgs(ll a, ll b, ll p, vector<ll>& ans) {

if (a % p == 0 && b) return;

map<ll, ll> hash;

b %= p;

ll t = sqrt(p) + 1;

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

hash[b \* qpow(a, i, p) % p] = i;

a = qpow(a, t, p);

if (!a) {

if (b) return;

else {

ans.push\_back(1);

return;

}

}

for (int i = 1; i <= t; i++) {

ll val = qpow(a, i, p);

int j = hash.find(val) == hash.end() ? -1 : hash[val];

if (j >= 0 && i \* t - j >= 0) ans.push\_back(i \* t - j);

}

}

inline ll generator(ll p){

if (p == 2) return 1;

vector<ll> fact;

ll phi = p - 1, n = phi;

for (int i = 2; i \* i <= n; i++)

if (n % i == 0) {

fact.push\_back(i);

while (n % i == 0)

n /= i;

}

if (n > 1) fact.push\_back(n);

for (int res = 2; res <= p; res++) {

bool ok = true;

for (ll factor : fact) {

if (qpow(res, phi / factor, p) == 1) {

ok = false;

break;

}

}

if (ok) return res;

}

return -1;

}

inline ll exbsgs(ll a, ll b, ll p) {

if (b == 1 || p == 1) return 0;

ll g = gcd(a, p), k = 0, na = 1;

while (g > 1) {

if (b % g != 0) return -1;

k++;

b /= g; p /= g;

na = na \* (a / g) % p;

if (na == b) return k;

g = gcd(a, p);

}

vector<ll> ans;

bsgs(a, b \* inv(na, p) % p, p, ans);

int f = ans.empty() ? -1 : ans[0];

if (f == -1) return -1;

return f + k;

}

//int main

if (!b){

printf("1\n0\n");

return;

}

#### //a^x=b(mod p)

bsgs(a, b, p); //a为底数,ans为指数,b为同余数

sort(all(ans));

#### //x^a=b(mod p)

vector<ll> ans1, ans2;

ll g = generator(p), gk = qpow(g, a, p);

bsgs(gk, b, p, ans1);

bsgs(g, b, p, ans2);

ll x0 = ans1.empty() ? -1 : ans1[0];

ll t = ans2.empty() ? -1 : ans2[0];

if (x0 == -1) {

printf("0\n");

return;

}

vector<ll> ans;

ll d = gcd(p - 1, a), mod = (p - 1) / d;

x0 = x0 % mod;

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

ans.push\_back(qpow(g, (x0 + i \* mod % (p - 1)) % (p - 1), p));

sort(all(ans));

#### //a^x=b(mod p),其中a,p不必互质

a %= p;

b %= p;

int t = exbsgs(a, b, p);

if (t == -1) printf("No Solution\n");

else printf("%lld\n", t);

## 高斯消元

#### 线性方程

ld a[M][M];

inline void GJ\_Elimination(int n) { //n个方程,n个项,1个结果

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

int max = i;

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

if (fabsl(a[j][i]) > fabsl(a[max][i])) max = j;

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

swap(a[i][j], a[max][j]);

if (!a[i][i]) {

puts("No Solution");

exit(0);

}

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

if (j != i) {

ld temp = a[j][i] / a[i][i];

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

a[j][k] -= a[i][k] \* temp;

}

}

}

//int main

ans[i] = a[i][n + 1] / a[i][i];

#### 行列式

inline ld Gauss\_Elimination(int n) { //n阶行列式

const ld EPS = 1e-9; //精度

vector<vector<ld> > a(n, vector<ld>(n));

ld det = 1;

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

int k = i;

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

if (fabsl(a[j][i]) > fabsl(a[k][i])) k = j;

if (fabsl(a[k][i]) < EPS) {

det = 0;

break;

}

swap(a[i], a[k]);

if (i != k) det = -det;

det \*= a[i][i];

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

a[i][j] /= a[i][i];

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

if (j != i && fabsl(a[j][i]) > EPS)

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

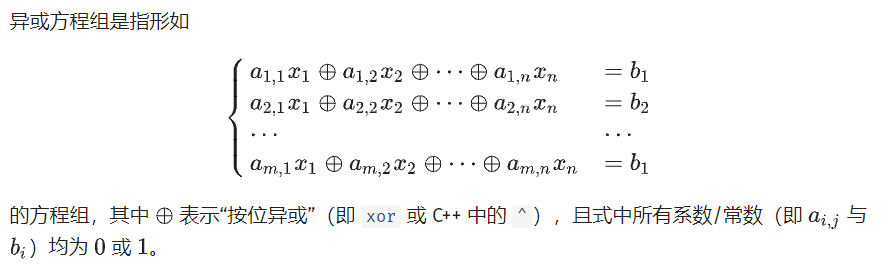
a[j][k] -= a[i][k] \* a[j][i];

}

return det;

}

#### 异或方程组



bitset<1010> matrix[2010]; //matrix[1~n]：增广矩阵,0位置为常数

inline vector<bool> Gauss\_Xor(int n, int m) { //n为未知数个数,m为方程个数

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

int cur = i;

while (cur <= m && !matrix[cur].test(i))

cur++;

if (cur > m) return vector<bool>(0);

if (cur != i) swap(matrix[cur], matrix[i]);

for (int j = 1; j <= m; j++)

if (i != j && matrix[j].test(i)) matrix[j] ^= matrix[i];

}

vector<bool> ans(n + 1, 0);

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

ans[i] = matrix[i].test(0);

return ans; //返回方程组的解(多解/无解返回一个空的 vector)

}

## 特征多项式

#include <random>

typedef vector<vector<int> > Matrix;

Matrix to\_upper\_Hessenberg(const Matrix& M, int mod) {

Matrix H(M);

int n = H.size();

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

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

if ((H[i][j] %= mod) < 0) H[i][j] += mod;

for (int i = 0; i < n - 1; i++) {

int pivot = i + 1;

for (; pivot < n; pivot++)

if (H[pivot][i]) break;

if (pivot == n) continue;

if (pivot != i + 1) {

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

Swap(H[i + 1][j], H[pivot][j]);

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

Swap(H[j][i + 1], H[j][pivot]);

}

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

while (H[j][i]) {

if (!H[i + 1][i]) {

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

Swap(H[i + 1][k], H[j][k]);

for (int k = 0; k < n; k++)

Swap(H[k][i + 1], H[k][j]);

break;

}

if (H[j][i] >= H[i + 1][i]) {

int q = H[j][i] / H[i + 1][i];

int mq = mod - q;

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

H[j][k] = (H[j][k] + (ll)mq \* H[i + 1][k]) % mod;

for (int k = 0; k < n; k++)

H[k][i + 1] = (H[k][i + 1] + (ll)q \* H[k][j]) % mod;

}

else {

int q = H[i + 1][i] / H[j][i];

int mq = mod - q;

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

H[i + 1][k] = (H[i + 1][k] + (ll)mq \* H[j][k]) % mod;

for (int k = 0; k < n; k++)

H[k][j] = (H[k][j] + (ll)q \* H[k][i + 1]) % mod;

}

}

}

return H;

}

inline vector<int> get\_charpoly(const Matrix& M, int mod) {

Matrix H(to\_upper\_Hessenberg(M, mod));

int n = H.size();

vector<vector<int> > p(n + 1);

p[0] = { 1 % mod };

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

const vector<int>& pi\_1 = p[i - 1];

vector<int>& pi = p[i];

pi.resize(i + 1, 0);

int v = mod - H[i - 1][i - 1];

if (v == mod) v -= mod;

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

pi[j] = (pi[j] + (ll)v \* pi\_1[j]) % mod;

if ((pi[j + 1] += pi\_1[j]) >= mod) pi[j + 1] -= mod;

}

int t = 1;

for (int j = 1; j < i; j++) {

t = (ll)t \* H[i - j][i - j - 1] % mod;

int prod = (ll)t \* H[i - j - 1][i - 1] % mod;

if (!prod) continue;

prod = mod - prod;

for (int k = 0; k <= i - j - 1; k++)

pi[k] = (pi[k] + (ll)prod \* p[i - j - 1][k]) % mod;

}

}

return p[n];

}

inline bool verify(const Matrix& M, const vector<int>& charpoly, int mod) {

if (mod == 1) return true;

int n = M.size();

vector<int> randvec(n), sum(n, 0);

mt19937 gen(random\_device{}());

uniform\_int\_distribution<int> dis(1, mod - 1);

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

randvec[i] = dis(gen);

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

int v = charpoly[i];

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

sum[j] = (sum[j] + (ll)v \* randvec[j]) % mod;

vector<int> prod(n, 0);

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

for (int k = 0; k < n; k++)

prod[j] = (prod[j] + (ll)M[j][k] \* randvec[k]) % mod;

randvec.swap(prod);

}

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

if (sum[i]) return false;

return true;

}

//int main

int n = read<int>();

int mod = read<int>();

Matrix M(n, vector<int>(n));

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

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

M[i][j] = read<int>();

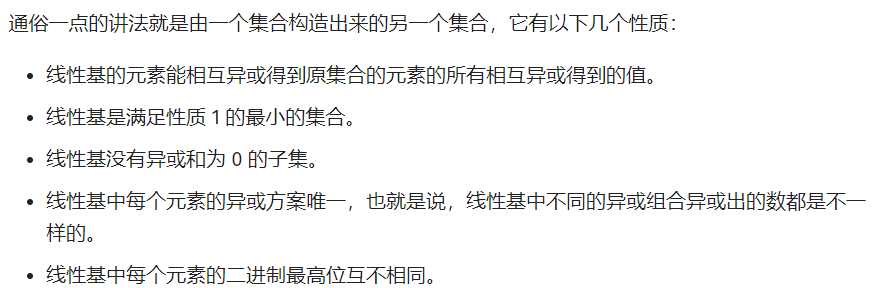
vector<int> charpoly(get\_charpoly(M, mod));

vector<int> ans;

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

ans.push\_back(charpoly[i]);

## 线性基



int p[M];

inline void insert(ll x) {

for (int i = 62; i + 1; i--) {

if (!(x >> i)) continue;

if (!p[i]) {

p[i] = x;

break;

}

x ^= p[i];

}

}

//int main

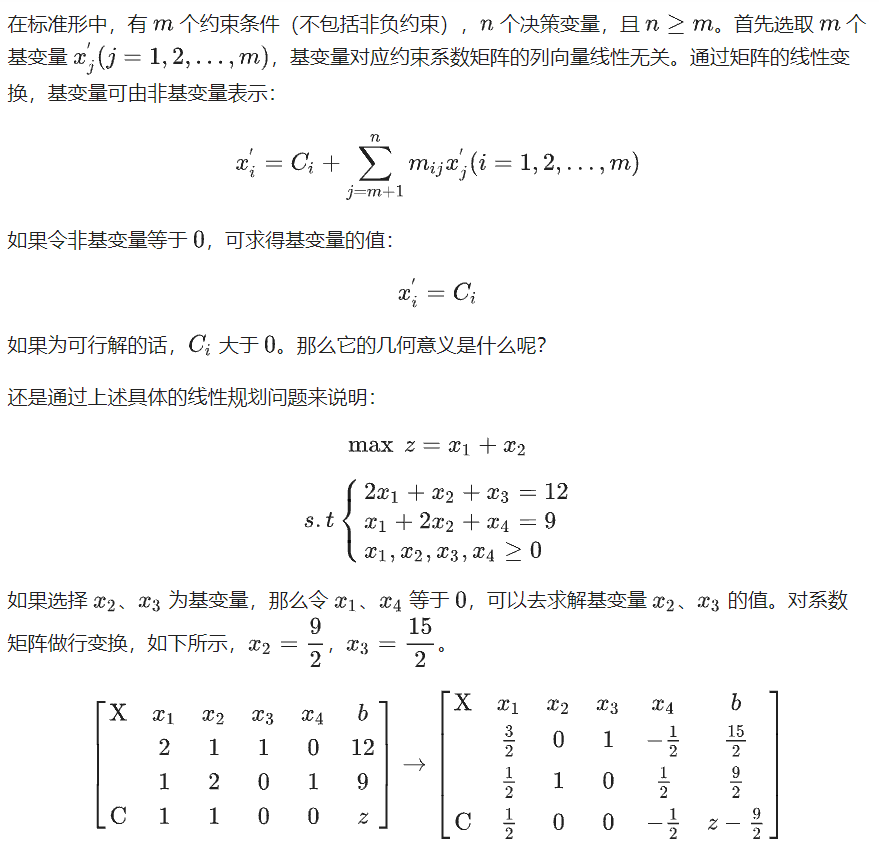
ll ans = 0;

for (int i = 62; i + 1; i--)

ans = max(ans, ans ^ p[i]);

## 线性规划

#### 单纯形法



vector<vector<ld> > Matrix;

ld Z;

set<int> P;

size\_t cn, bn;

inline bool Pivot(pair<size\_t, size\_t>& p) { //返回false表示所有的非轴元素都小于0

int x = 0, y = 0;

ld cmax = -INT\_MAX;

vector<ld> C = Matrix[0];

vector<ld> B;

for (size\_t i = 0; i < bn; i++)

B.push\_back(Matrix[i][cn - 1]);

for (size\_t i = 0; i < C.size(); i++)

if (cmax < C[i] && P.find(i) == P.end()) {

cmax = C[i];

y = i;

}

if (cmax < 0) return false;

ld bmin = INT\_MAX;

for (size\_t i = 1; i < bn; i++) {

ld tmp = B[i] / Matrix[i][y];

if (Matrix[i][y] && bmin > tmp) {

bmin = tmp;

x = i;

}

}

p = make\_pair(x, y);

for (auto it = P.begin(); it != P.end(); it++)

if (Matrix[x][\*it]) {

P.erase(\*it);

break;

}

P.insert(y);

return true;

}

inline void print() {

for (size\_t i = 0; i < Matrix.size(); i++) {

for (size\_t j = 0; j < Matrix[0].size(); j++)

cout << Matrix[i][j] << "\t";

pn;

}

printf("result z:%d\n", -Matrix[0][cn - 1]);

}

inline void Gaussian(pair<size\_t, size\_t> p) {

size\_t x = p.first;

size\_t y = p.second;

ld norm = Matrix[x][y];

for (size\_t i = 0; i < cn; i++)

Matrix[x][i] /= norm;

for (size\_t i = 0; i < bn; i++)

if (i != x && Matrix[i][y]) {

ld tmpnorm = Matrix[i][y];

for (size\_t j = 0; j < cn; j++)

Matrix[i][j] = Matrix[i][j] - tmpnorm \* Matrix[x][j];

}

}

//int main

cn = read<int>();

bn = read<int>();

for (size\_t i = 0; i < bn; i++) {

vector<ld> vectmp;

for (size\_t j = 0; j < cn; j++) {

ld tmp = 0;

scanf("%Lf", &tmp);

vectmp.push\_back(tmp);

}

Matrix.push\_back(vectmp);

}

for (size\_t i = 0; i < bn - 1; i++)

P.insert(cn - i - 2);

pair<size\_t, size\_t> t;

while (1) {

pnt();

if (!Pivot(t)) return;

printf("%d %d\n", t.first, t.second);

for (auto it = P.begin(); it != P.end(); it++)

cout << \*it << " ";

pn;

Gaussian(t);

}

#### 理论罗列

int n, m;

ld v;

ld a[N][M], b[N], c[M];

inline void pivot(int l, int e) { //转轴

b[l] /= a[l][e];

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

if (j != e) a[l][j] /= a[l][e];

a[l][e] = 1 / a[l][e];

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

if (i != l && fabsl(a[i][e]) > 0) {

b[i] -= a[i][e] \* b[l];

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

if (j != e) a[i][j] -= a[i][e] \* a[l][j];

a[i][e] = -a[i][e] \* a[l][e];

}

v += c[e] \* b[l];

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

if (j != e) c[j] -= c[e] \* a[l][j];

c[e] = -c[e] \* a[l][e];

}

inline ld simplex() {

while (1) {

int e = 0, l = 0;

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

if (c[e] > (ld)0) break;

if (e == n + 1) return v; //此时v即为最优解

ld mn = INF;

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

if (a[i][e] > (ld)0 && mn > b[i] / a[i][e]) {

mn = b[i] / a[i][e];

l = i;

}

if (mn == INF) return INF;

pivot(l, e);

}

}

//int main

n = read<int>();

m = read<int>();

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

c[i] = read<int>();

for (int i = 1; i <= m; i++) {

int s = read<int>(), t = read<int>();

for (int j = s; j <= t; j++)

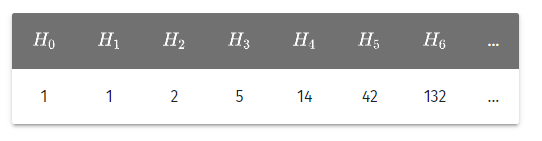
a[i][j] = 1;

b[i] = read<int>();

}

int ans = simplex() + 0.5;

## 卡特兰数



int H[N]; //卡特兰数

void init(int n) { //初始化卡特兰数

H[0] = 1;

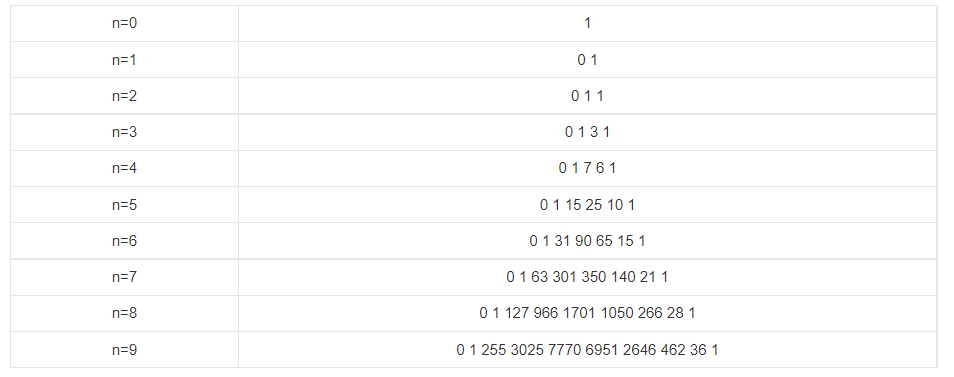
for (int i = 1; i <= n; i++)

H[i] = H[i - 1] \* (4 \* i - 2) / (i + 1);

}

## 斯特林数

#### 第二类



###### 同一列O(nlogn)

int n, k;

ll g[N << 2], r[N << 2], p[N << 2];

ll f[N << 2], s[N << 2], stirling2C[N << 2];

int G = 3;

ll invn, invG;

ll fac[N], inv[N];

inline ll qpow(ll a, ll b = mod – 2, ll p = mod) {

ll ans = 1;

while (b) {

if (b & 1) ans = (ans \* a) % p;

a = (a \* a) % p;

b >>= 1;

}

return ans;

}

inline void NTT(ll\* f, bool op, int n) {

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

if (r[i] < i) swap(f[r[i]], f[i]);

for (int len = 1; len < n; len <<= 1) {

int w = qpow(op == 1 ? G : invG, (mod - 1) / len / 2);

for (int p = 0; p < n; p += len + len) {

ll buf = 1;

for (int i = p; i < p + len; i++) {

int sav = f[i + len] \* buf % mod;

f[i + len] = f[i] - sav;

if (f[i + len] < 0) f[i + len] += mod;

f[i] = f[i] + sav;

if (f[i] >= mod) f[i] -= mod;

buf = buf \* w % mod;

}

}

}

}

inline void rev(ll\* f, int len) {

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

g[i] = f[i];

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

f[len - i - 1] = g[i];

}

inline void times(ll\* f, ll\* gg, int len, int lim) {

int m = len + len, n;

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

g[i] = gg[i];

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

invn = qpow(n);

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

g[i] = 0;

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

r[i] = (r[i >> 1] >> 1) | ((i & 1) ? n >> 1 : 0);

NTT(f, 1, n);

NTT(g, 1, n);

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

f[i] = (f[i] \* g[i]) % mod;

NTT(f, 0, n);

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

f[i] = (f[i] \* invn) % mod;

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

f[i] = 0;

}

inline void Init(int lim) {

inv[1] = inv[0] = fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

for (int i = 2; i <= lim; i++)

inv[i] = inv[mod % i] \* (mod - mod / i) % mod;

for (int i = 2; i <= lim; i++)

inv[i] = inv[i - 1] \* inv[i] % mod;

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

inv[i] = qpow(fac[i]);

}

inline void fminus(ll\* s, ll\* f, int len, int c) {

c = mod - c;

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

p[len - i - 1] = f[i] \* fac[i] % mod;

ll buf = 1;

for (int i = 0; i < len; i++, buf = buf \* c % mod)

s[i] = buf \* inv[i] % mod;

times(p, s, len, len);

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

s[len - i - 1] = p[i] \* inv[len - i - 1] % mod;

for (int i = len; i < len + len; i++)

s[i] = 0;

}

inline void Solve(ll\* f, int n) {

if (n == 1) {

f[0] = 0;

f[1] = 1;

}

else if (n & 1) {

Solve(f, n - 1);

f[n] = 0;

for (int i = n; i > 0; i--)

f[i] = (f[i - 1] + (mod - n + 1) \* f[i]) % mod;

f[0] = f[0] \* (mod - n + 1) % mod;

}

else {

Solve(f, n / 2);

fminus(s, f, n / 2 + 1, n / 2);

times(f, s, n / 2 + 1, n + 1);

}

}

inline void invp(ll\* f, int len) {

for (int i = 0; i < k + 1; i++)

s[i] = p[i] = 0;

ll\* r = s, \* rr = p;

int n = 1;

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

rr[0] = qpow(f[0]);

for (int len = 2; len <= n; len <<= 1) {

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

r[i] = rr[i] \* 2 % mod;

times(rr, rr, len / 2, len);

times(rr, f, len, len);

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

rr[i] = (r[i] - rr[i] + mod) % mod;

}

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

f[i] = rr[i];

}

//int main

int n = read<int>();

int k = read<int>();

if (k > n) {

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

printf("0 ");

return;

}

invG = qpow(G);

Init(k);

Solve(f, k + 1);

for (int i = 0; i < k + 1; i++)

f[i] = f[i + 1];

rev(f, k + 1);

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

f[i] = 0;

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

f[i] = 0;

invp(f, n - k + 1);

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

stirling2C[i] = 0;

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

stirling2C[i] = f[i - k];

###### 同一行O(nlogn)

using namespace Poly;

ll fac[N], ifac[N];

inline void exgcd(ll a, ll b, ll& x, ll& y) {

if (!b) {

x = 1;

y = 0;

return;

}

exgcd(b, a % b, y, x);

y -= (a / b) \* x;

}

inline void fact(ll n) { //mod可不为质数

fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

exgcd(fac[n], mod, ifac[n], ifac[0]);

ifac[n] = (ifac[n] % mod + mod) % mod;

for (int i = n - 1; i >= 0; i--)

ifac[i] = ifac[i + 1] \* (i + 1) % mod;

}

//int main

int n = read<int>();

fact(n);

poly f(n + 1), g(n + 1), stirling2R(n + 1);

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

f[i] = (i & 1 ? mod - ifac[i] : ifac[i]) % mod;

g[i] = qpow(i, n) \* ifac[i] % mod;

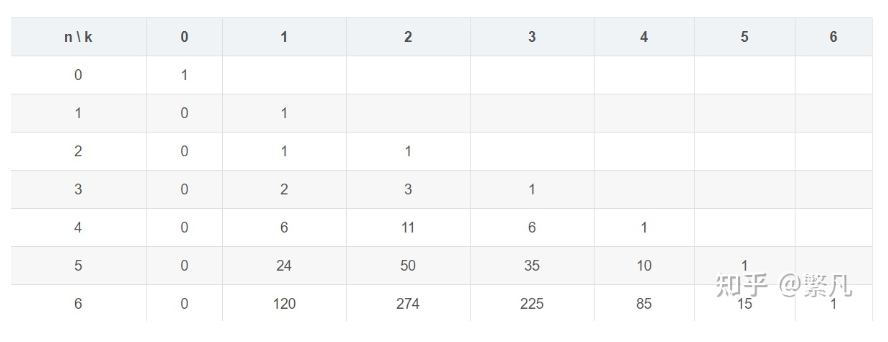
}

f = f \* g;

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

stirling2R[i] = f[i];

#### 第一类



###### 同一列O(nlogn)

using namespace Poly;

ll fac[N], ifac[N], stirling1C[N];

inline void fact(ll n) { //mod需为质数

fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

ifac[n] = qpow(fac[n], mod - 2);

for (int i = n; i >= 1; i--)

ifac[i - 1] = ifac[i] \* i % mod;

}

//int main

int n, k;

read(n, k);

poly A(n), B;

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

A[i] = qpow(i + 1, mod - 2);

A = polyLn(A, n);

A = A \* k;

B = polyExp(A, n);

fact(n);

B.resize(n + 1);

B.insert(B.begin(), k, 0);

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

stirling1C[i] = 1ll \* B[i] \* fac[i] % mod \* ifac[k] % mod;

###### 同一行O(nlogn)

ll G = 3, invG;

int R[N];

ll inv[N], fac[N];

ll w[N], a[N], b[N], g[N];

ll f[N], stirling1R[N];

inline ll qpow(ll a, ll b, ll p = mod) {

ll res = 1;

while (b) {

if (b & 1) res = res \* a % p;

a = a \* a % p;

b >>= 1;

}

return res;

}

inline void NTT(ll\* f, int n, int fl) {

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

if (i < R[i]) swap(f[i], f[R[i]]);

for (int p = 2; p <= n; p <<= 1) {

int len = (p >> 1);

ll w = qpow((fl == 0) ? G : invG, (mod - 1) / p);

for (int st = 0; st < n; st += p) {

ll buf = 1, tmp;

for (int i = st; i < st + len; i++) {

tmp = buf \* f[i + len] % mod;

f[i + len] = (f[i] - tmp + mod) % mod;

f[i] = (f[i] + tmp) % mod;

buf = buf \* w % mod;

}

}

}

if (fl == 1) {

ll invN = qpow(n, mod - 2);

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

f[i] = (f[i] \* invN) % mod;

}

}

inline void Mull(ll\* f, ll\* g, int n, int m) {

m += n;

n = 1;

while (n < m)

n <<= 1;

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

R[i] = (R[i >> 1] >> 1) | ((i & 1) ? (n >> 1) : 0);

NTT(f, n, 0);

NTT(g, n, 0);

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

f[i] = f[i] \* g[i] % mod;

NTT(f, n, 1);

}

inline void Solve(ll\* f, int m) {

if (m == 1) {

f[1] = 1;

return;

}

if (m & 1) {

Solve(f, m - 1);

for (int i = m; i >= 1; i--)

f[i] = (f[i - 1] + f[i] \* (m - 1) % mod) % mod;

f[0] = f[0] \* (m - 1) % mod;

}

else {

int n = m / 2;

ll res = 1;

Solve(f, n);

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

a[i] = f[i] \* fac[i] % mod;

b[i] = res \* inv[i] % mod;

res = res \* n % mod;

}

reverse(a, a + n + 1);

Mull(a, b, n + 1, n + 1);

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

g[i] = inv[i] \* a[n - i] % mod;

Mull(f, g, n + 1, n + 1);

int limit = 1;

while (limit < (n + 1) << 1)

limit <<= 1;

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

a[i] = b[i] = g[i] = 0;

for (int i = m + 1; i < limit; i++)

f[i] = 0;

}

}

inline void init(int n) {

fac[0] = 1;

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

fac[i] = 1ll \* fac[i - 1] \* i % mod;

inv[n] = qpow(fac[n], mod - 2);

for (int i = n - 1; i >= 0; i--)

inv[i] = 1ll \* inv[i + 1] \* (i + 1) % mod;

}

//int main

invG = qpow(G, mod - 2);

int k = 0;

int n = read<int>();

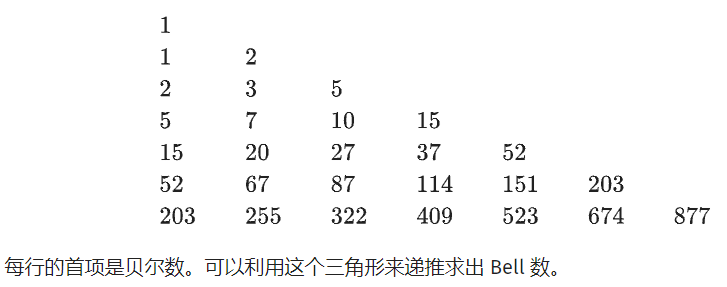
init(n + n);

Solve(f, n);

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

stirling1R[i] = f[i];

## 贝尔数



int bell[N][N];

inline void Bell(int n) { //每行首项为bell数

bell[1][1] = 1;

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

bell[i][1] = bell[i - 1][i - 1];

for (int j = 2; j <= i; j++)

bell[i][j] = bell[i - 1][j - 1] + bell[i][j - 1];

}

}

或

using namespace Poly;

inline poly initBell(int n) {

fact(n);

poly a(n + 1), bell;

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

a[i] = ifac[i];

bell = polyExp(a, n + 1);

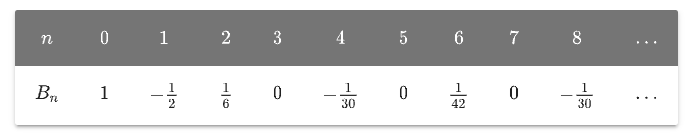
for (int i = 0; i <= n; i++)

bell[i] = mul(bell[i], fac[i]);

return bell;

}

## 伯努利数



ll B[N]; // 伯努利数

ll fac[N], ifac[N];

inline void init() { //预处理伯努利数

fact(N - 1);

B[0] = 1;

for (int i = 1; i < N; i++) {

ll ans = 0;

if (i == N - 1) break;

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

ans += C(i + 1, k) \* B[k];

ans %= mod;

}

ans = (ans \* (-ifac[i + 1]) % mod + mod) % mod;

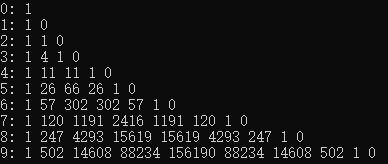
B[i] = ans;

}

}

## 欧拉数

#### 一阶欧拉数



using namespace Poly;

ll fac[N], ifac[N];

inline void fact(ll n) { //mod需为质数

fac[0] = 1;

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

fac[i] = fac[i - 1] \* i % mod;

ifac[n] = qpow(fac[n], mod - 2);

for (int i = n; i >= 1; i--)

ifac[i - 1] = ifac[i] \* i % mod;

}

//int main

int n = read<int>();

fact(n + 1);

poly f(n + 1), g(n + 1), eulerian(n + 1);

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

g[i] = qpow(i + 1, n);

f[i] = fac[n + 1] \* ifac[i] % mod \* ifac[n + 1 - i] % mod \* (i & 1 ? mod - 1 : 1) % mod;

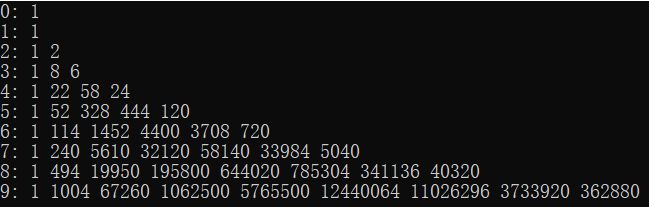
}

f = f \* g;

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

eulerian[i] = f[i];

#### 二阶欧拉数



int Euler2[N][N];

inline void initEuler2(int p) {

Euler2[0][0] = 1;

for (int n = 1; n < N; n++) {

Euler2[n][0] = 1;

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

Euler2[n][k] = ((2ll \* n - k - 1) \* Euler2[n - 1][k - 1] + (k + 1ll) \* Euler2[n - 1][k]) % mod;

}

}

## 康托展开

#### 康托展开

int n;

int a[N];

ll A[N] = { 1,1 };

ll tree[N]; //树状数组

inline void update(int x, int y) {

while (x <= n) {

tree[x] += y;

x += lowbit(x);

}

}

inline ll query(int x) {

ll sum = 0;

while (x) {

sum += tree[x];

x -= lowbit(x);

}

return sum;

}

//int main

int n = read<int>();

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

A[i] = (A[i - 1] \* i) % mod;

update(i, 1);

}

ll ans = 0;

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

a[i] = read<int>();

ans = (ans + ((query(a[i]) - 1) \* A[n - i]) % mod) % mod;

update(a[i], -1);

}

ans++;

#### 逆康托展开

int tr[N << 2];

inline void Build(int c, int l, int r) {

if (l == r) {

tr[c] = 1;

return;

}

int mid((l + r) >> 1), ls(c << 1), rs(c << 1 | 1);

Build(ls, l, mid);

Build(rs, mid + 1, r);

tr[c] = tr[ls] + tr[rs];

}

inline int Get(int c, int l, int r, int k) {

tr[c]--;

if (l == r) return l;

int mid((l + r) >> 1), ls(c << 1), rs(ls | 1);

if (tr[ls] < k) return Get(rs, mid + 1, r, k - tr[ls]);

return Get(ls, l, mid, k);

}

//int main

int n = read<int>();

Build(1, 1, n);

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

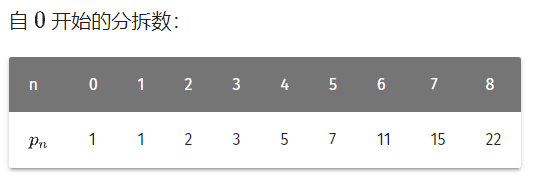
int s = read<int>();

int ans = Get(1, 1, n, s + 1);

}

## 分拆数

#### 分拆数



ll a[N];

ll p[N]; //分拆数

inline void init(int n) { //初始化分拆数

p[0] = p[1] = 1;

p[2] = 2;

for (int i = 1; i < N; i++) {

a[2 \* i] = i \* (i \* 3 - 1) / 2;

a[2 \* i + 1] = i \* (i \* 3 + 1) / 2;

}

for (int i = 3; i < N; i++) {

p[i] = 0;

for (int j = 2; a[j] <= i; j++) {

if (j & 2) p[i] = (p[i] + p[i - a[j]] + mod) % mod;

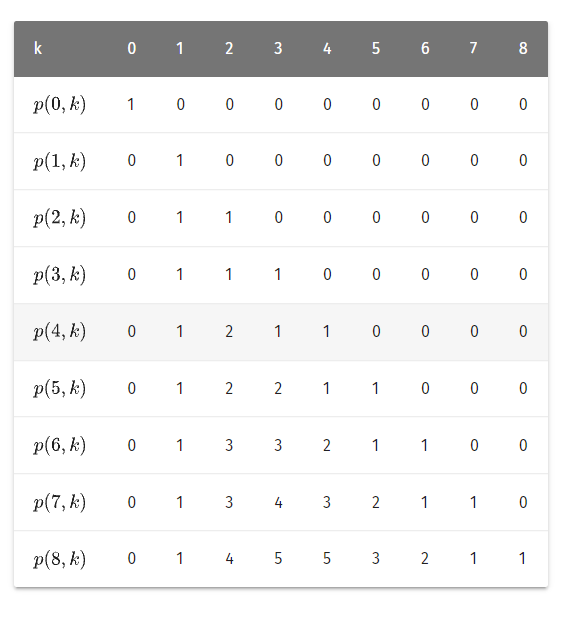
else p[i] = (p[i] - p[i - a[j]] + mod) % mod;

}

}

}

#### K部分拆数



int p[N][M]; //k部分拆数

inline void init(int n, int k) { //初始化k部分拆数

memset(p, 0, sizeof(p));

p[0][0] = 1;

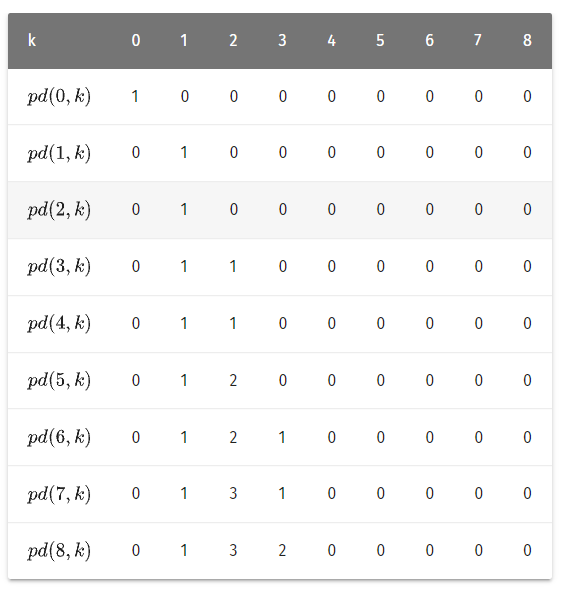
for (int i = 1; i <= n; i++)

for (int j = 1; j <= k; j++)

if (i - j >= 0) p[i][j] = (p[i - j][j] + p[i - 1][j - 1]) % mod;

}

#### 互异分拆数



ll pd[N][2]; //互异分拆数

inline void init(int n) { //初始化互异分拆数

memset(pd, 0, sizeof(pd));

pd[0][0] = 1;

int ans = 0;

for (int j = 1; j < M; j++) {

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

pd[i][j & 1] = 0;

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

if (i - j >= 0) pd[i][j & 1] = (pd[i - j][j & 1] + pd[i - j][(j - 1) & 1]) % mod;

}

}

//int main

for (int i = 1; i < M; i++) //M一般为20\*logn

ans = (ans + pd[x][i & 1]) % mod; //计算pdx

## 斐波那契

#### 矩阵快速幂求斐波那契

struct Matrix {

ll val[3][3];

inline Matrix operator \* (const Matrix& a) const {

Matrix Res = (\*this);

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

for (int j = 1; j <= 2; j++)

Res.val[i][j] = (val[i][1] \* a.val[1][j] % mod + val[i][2] \* a.val[2][j] % mod) % mod;

return Res;

}

inline Matrix operator \*= (const Matrix& a) {

return (\*this) = a \* (\*this);

}

inline Matrix operator ^ (ll k) const {

Matrix Res, Temp = (\*this);

Res.val[1][1] = Res.val[2][2] = 1;

Res.val[1][2] = Res.val[2][1] = 0;

while (k) {

if (k & 1) Res \*= Temp;

k >>= 1, Temp \*= Temp;

}

return Res;

}

} Fibo, Cell;

inline void calc(ll n) {

if (!n) {

printf("0");

return;

}

Fibo.val[1][1] = 0;

Fibo.val[2][1] = Fibo.val[1][2] = Fibo.val[2][2] = 1;

Cell.val[1][1] = 1, Cell.val[1][2] = 1;

Matrix Res = Cell \* (Fibo ^ (n - 1));

printf("%lld\n", Res.val[1][1] % mod);

}

//int main

ll n = read<int>();

calc(n);

#### 快速查找两个相邻fib

inline pii fib(int n) { //返回fn和fn+1

if (!n) return { 0, 1 };

auto p = fib(n >> 1);

int c = p.first \* (2 \* p.second - p.first);

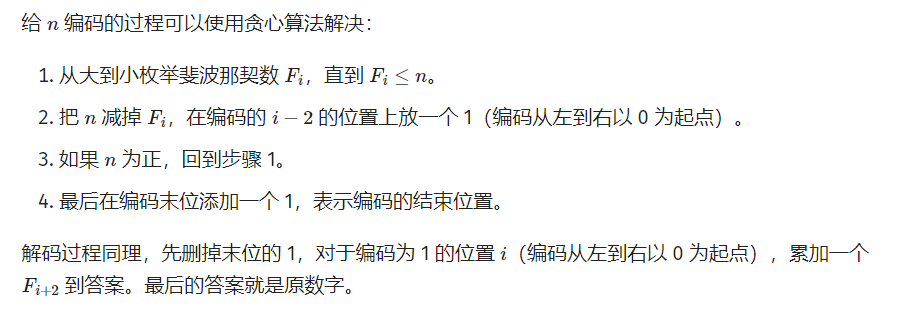
int d = p.first \* p.first + p.second \* p.second;

if (n & 1) return { d, c + d };

else return { c, d };

}

#### 斐波那契编码



#### 斐波那契取模

int P;

int Fac[30], Tim[30], tot;

char ch[N];

struct Matrix {

ll val[3][3];

inline Matrix operator \* (const Matrix& a) const {

Matrix Res = (\*this);

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

for (int j = 1; j <= 2; j++)

Res.val[i][j] = (val[i][1] \* a.val[1][j] % P + val[i][2] \* a.val[2][j] % P) % P;

return Res;

}

inline Matrix operator \*= (const Matrix& a) {

return (\*this) = a \* (\*this);

}

inline Matrix operator ^ (ll k) const {

Matrix Res, Temp = (\*this);

Res.val[1][1] = Res.val[2][2] = 1;

Res.val[1][2] = Res.val[2][1] = 0;

while (k) {

if (k & 1) Res \*= Temp;

k >>= 1, Temp \*= Temp;

}

return Res;

}

} Fibo, Cell;

inline ll qpow(ll a, ll b, ll p = mod) {

ll res = 1;

while (b) {

if (b & 1) res = res \* a % p;

a = a \* a % p;

b >>= 1;

}

return res;

}

inline void Divide(int p) { //分解质因数

for (int i = 2; i \* i <= p; i++)

if (p % i == 0) {

Fac[++tot] = i;

while (p % i == 0) {

p /= i;

Tim[tot]++;

}

}

if (p != 1) {

Fac[++tot] = p;

Tim[tot] = 1;

}

}

inline ll PrimeLoop(ll p) { //求pi(p)

if (p == 2) return 3;

if (p == 5) return 20;

if (qpow(5, (p - 1) >> 1, p) == 1) return p - 1;

return 2 \* p + 2;

}

inline ll PrimePow(int id) { //求pi(p^k)

return qpow(Fac[id], Tim[id] - 1) \* PrimeLoop(Fac[id]);

}

inline ll GetLoop() { //求pi(p)

ll res = PrimePow(1), temp;

for (int i = 2; i <= tot; i++) {

temp = PrimePow(i);

res = res / gcd(res, temp) \* temp;

}

return res;

}

inline void calc(ll n) {

if (!n) {

printf("0");

return;

}

Fibo.val[1][1] = 0;

Fibo.val[2][1] = Fibo.val[1][2] = Fibo.val[2][2] = 1;

Cell.val[1][1] = 1, Cell.val[1][2] = 1;

Matrix Res = Cell \* (Fibo ^ (n - 1));

printf("%lld\n", Res.val[1][1] % P);

}

//int main

scanf("%s", ch);

P = read<int>();

if (P == 1) {

printf("0");

return;

}

Divide(P);

ll Loop = GetLoop(), n = 0;

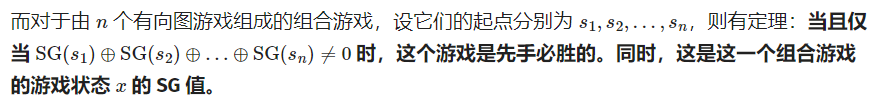
int Len = strlen(ch);

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

n = (n \* 10 + (ch[i] ^ 48)) % Loop;

calc(n);

## SG函数



int f[M], sg[M], mex[M];

inline int getSG(int n) {

sg[0] = 0;

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

memset(mex, 0, sizeof(mex));

for (int j = 1; f[j] <= i; j++)

mex[sg[i - f[j]]] = 1;

for (int j = 0;; j++)

if (!mex[j]) {

sg[i] = j;

break;

}

}

return sg[n];

}

## 平方根

#### 根据精度求解

inline ld Sqrt(ld n) {

const ld eps = 1E-15; //手动设置精度

ld x = 1;

while (1) {

ld nx = (x + n / x) / 2;

if (fabsl(x - nx) < eps) break;

x = nx;

}

return x;

}

#### 求解最大整数

inline ll isqrt(ll n) {

ll x = 1;

bool decreased = false;

while (1) {

ll nx = (x + n / x) >> 1;

if (x == nx || (nx > x && decreased)) break;

decreased = nx < x;

x = nx;

}

return x;

}

## 求解积分方程

inline ld f(ld x) {

}

inline ld simpson(ld l, ld r) {

ld mid = (l + r) / 2;

return (r - l) \* (f(l) + 4 \* f(mid) + f(r)) / 6;

}

inline ld asr(ld l, ld r, ld eqs, ld ans, int step) {

ld mid = (l + r) / 2;

ld fl = simpson(l, mid), fr = simpson(mid, r);

if (fabsl(fl + fr - ans) <= 15 \* eqs && step < 0)

return fl + fr + (fl + fr - ans) / 15;

return asr(l, mid, eqs / 2, fl, step - 1) + asr(mid, r, eqs / 2, fr, step - 1);

}

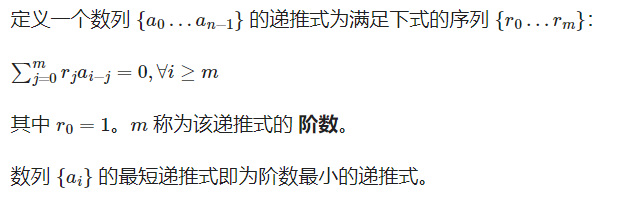
inline ld calc(ld l, ld r, ld eps) {

return asr(l, r, eps, simpson(l, r), 12);

}

## BM

#### 求解阶数最小的递推式



inline vector<int> BM(const vector<int>& a) {

vector<int> v, last;

int k = -1, delta = 0;

int n = a.size();

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

int tmp = 0;

int m = v.size();

for (int j = 0; j < m; j++)

tmp = (tmp + (ll)a[i - j - 1] \* v[j]) % mod;

if (a[i] == tmp) continue;

if (k < 0) {

k = i;

delta = (a[i] - tmp + mod) % mod;

v = vector<int>(i + 1);

continue;

}

vector<int> u = v;

int val = (ll)(a[i] - tmp + mod) \* qpow(delta, mod - 2) % mod;

if (v.size() < last.size() + i - k) v.resize(last.size() + i - k);

(v[i - k - 1] += val) %= mod;

int m = last.size();

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

v[i - k + j] = (v[i - k + j] - (ll)val \* last[j]) % mod;

if (v[i - k + j] < 0) v[i - k + j] += mod;

}

if ((int)u.size() - i < (int)last.size() - k) {

last = u;

k = i;

delta = a[i] - tmp;

if (delta < 0) delta += mod;

}

}

for (auto& x : v)

x = (mod - x) % mod;

v.insert(v.begin(), 1);

return v;

}

#### 求解稀疏方程组

#include <random>

struct node {

int x, y, value;

};

inline vector<int> solve\_sparse\_equations(const vector<node>& A, const vector<int>& b) {

int n = (int)b.size();

vector<vector<int> > f({ b });

for (int i = 1; i < 2 \* n; i++) {

vector<int> v(n);

auto& u = f.back();

for (auto it : A)

v[it.x] = (v[it.y] + (ll)u[it.y] \* it.value) % mod;

f.push\_back(v);

}

vector<int> w(n);

mt19937 gen;

for (auto& x : w)

x = uniform\_int\_distribution<int>(1, mod - 1)(gen);

vector<int> a(2 \* n);

for (int i = 0; i < 2 \* n; i++)

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

a[i] = (a[i] + (ll)f[i][j] \* w[j]) % mod;

auto c = BM(a);

int m = (int)c.size();

vector<int> ans(n);

for (int i = 0; i < m - 1; i++)

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

ans[j] = (ans[j] + (ll)c[m - 2 - i] \* f[i][j]) % mod;

int inv = qpow(mod - c[m - 1], mod - 2);

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

ans[i] = (ll)ans[i] \* inv % mod;

return ans;

}

## Powerful Number

int cnt;

int fac[N], prime[N];

ll inv[64];

inline ll mul(ll a, ll b, ll p = mod) {

return (a \* b - (ull)((ld)a / p \* b) \* p + p) % p;

}

inline ll cal(int k, ll pk) {

return mod - mul(pk, mul(inv[k], inv[k - 1]));

}

inline ll qpow(ll a, ll b, ll p = mod) {

ll res = 1;

while (b) {

if (b & 1) res = mul(res, a);

a = mul(a, a);

b >>= 1;

}

return res;

}

inline void primes() {

for (int i = 2; i <= M; i++) {

if (!fac[i]) fac[i] = prime[++cnt] = i;

for (int j = 1; j <= cnt; j++)

if (prime[j] > fac[i] || prime[j] \* i > M) break;

else fac[prime[j] \* i] = prime[j];

}

}

inline void init() {

primes();

for (int i = 1; i < 64; i++)

inv[i] = qpow(i, mod - 2);

}

inline ll pn(ll n, int l, ll r) {

ll res = mul(mul(n, mul(n + 1, inv[2])), r);

for (int i = l + 1; i <= cnt; i++) {

int pr = prime[i], k = 1;

ll val = n / pr, pk = pr;

if (val < pr) break;

while (val >= pr) {

val /= pr;

pk \*= pr;

k++;

res += pn(val, i, mul(r, cal(k, pk)));

if (res >= mod) res -= mod;

}

}

return res;

}

## Min\_25筛(常数小，1s能跑1e10)

const int mod = 1e9 + 7;

const int inv2 = 5e8 + 4;

const int inv6 = 166666668;

bool isp[N];

ll n, m, sz, sqrtN, c0, c1;

ll p[N], w[N], id0[N], id1[N], g0[N], g1[N], sum0[N], sum1[N];

//n是输入的数, sqrt\_n保存sqrt(n)，即预处理的数量

//sz是质数个数, isp[i]表示i是否质数, p[i]存储第i个质数

//sum0[i]存储的是前i个质数的f0值之和,sum1[i]存储的是前i个质数的f1值之和

//m是n/k的个数, w[i]存储n/k第i种值(倒序), id0和id1[i]存储i这个值在w[i]中的下标

//g0[i]和g1[i]等分别存储f在取质数时的多项式中的不同次方项(此处只有两个数组，即假设题目中的f取质数时只有两项), g0[i]存的是g(w[i],0~sz), g1[i]存的是g1(w[i],0~sz)

//g0(n,i) = \Sigma\_{j=1}^{n}[j是质数 or j的最小质因子>p[i]]\*f0(j) 其中f0为f取质数时的第一个次方项

//g1(n,i) = \Sigma\_{j=1}^{n}[j是质数 or j的最小质因子>p[i]]\*f1(j) 其中f1为f取质数时的第二个次方项

//c0和c1等保存的是不同次方项的系数(此处只有两个系数，即假设题目中的f取质数时只有两项)

//计算f(p^t)，若要降低常数也可把这个函数用增量法在调用处实现

inline ll f(ll p, ll t) {

//...

}

//线性筛，求函数f0、f1在前i个质数处的前缀和

inline void init(ll n) {

sz = 0;

memset(isp, 1, sizeof(isp));

isp[1] = 0;

sum0[0] = 0;

sum1[0] = 0;

for (ll i = 2; i <= n; i++) {

if (isp[i]) {

p[++sz] = i;

//计算sum0，即sum0(i) = \Sigma\_{j=1}^{i}f0(p[j])

//...

//计算sum1，即sum1(i) = \Sigma\_{j=1}^{i}f1(p[j])

//...

}

for (int j = 1; j <= sz && p[j] \* i <= n; j++) {

isp[i \* p[j]] = 0;

if (i % p[j] == 0) break;

}

}

}

inline int getId(ll x) {

if (x <= sqrtN) return id0[x];

else return id1[n / x];

}

//计算原理中的多项式的项, 只会计算g0(n/i), g1(n/i)

inline void sieveG(ll n) {

m = 0;

for (ll i = 1, j; i <= n; i = j + 1) {

ll k = n / i;

j = n / k;

w[++m] = k;

if (k <= sqrtN) id0[k] = m;

else id1[n / k] = m;

k %= mod;

//计算原理中的g0(w[m],0)，即\Sigma\_{j=2}^{w[m]}f0(j), 存在g0[m]中

//...

//计算原理中的g1(w[m],0)，即\Sigma\_{j=2}^{w[m]}f1(j), 存在g1[m]中

//...

}

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

for (int j = 1; j <= m && p[i] \* p[i] <= w[j]; j++) {

int op = getId(w[j] / p[i]);

//根据g0[j]=(g0[j]-f0(p[i])\*((g0[op]-sum0[i-1]+MOD)%MOD)%MOD+MOD)%MOD计算

//...

//根据g1[j]=(g1[j]-f1(p[i])\*((g1[op]-sum1[i-1]+MOD)%MOD)%MOD+MOD)%MOD计算

//...

}

}

inline ll S(ll x, ll y) {

if (x <= 1 || p[y] > x) return 0;

ll k = getId(x), res = 0;

res = ((c0 \* g0[k] % mod + c1 \* g1[k] % mod + mod) % mod -

(c0 \* sum0[y - 1] % mod + c1 \* sum1[y - 1] % mod + mod) % mod + mod) % mod;

for (int i = y; i <= sz && p[i] \* p[i] <= x; i++) {

ll t0 = p[i], t1 = p[i] \* p[i];

for (ll e = 1; t1 <= x; t0 = t1, t1 \*= p[i], e++) {

ll fp0 = f(p[i], e), fp1 = f(p[i], e + 1);

(res += (fp0 \* S(x / t0, i + 1) % mod + fp1) % mod) %= mod;

}

}

return res;

}

//int main

sqrtN = sqrt(n);

init(sqrtN);

sieveG(n);

//此处对不同次项的系数c0,c1进行直接赋值

//...

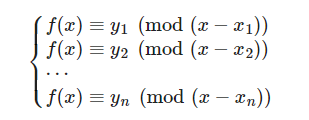
//此处计算的是原函数f在取值为1时的函数值，即f(1)，存在f\_1中；若是积性函数的话一般有f(1)=1

//...

ll f1 = 1;

ll ans = ((S(n, 1) + f1) % mod);

## 拉格朗日插值



//O(n^2)求解f(k)

int x[N], y[N];

//int main

int n = read<int>();

int k = read<int>();

int s1, s2, ans = 0;

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

s1 = y[i] % mod;

s2 = 1;

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

if (i != j) {

s1 = s1 \* (k - x[j]) % mod;

s2 = s2 \* (x[i] - x[j]) % mod;

}

ans += s1 \* qpow(s2, mod - 2) % mod;

}

ans = (ans % mod + mod) % mod;

## ST表

ll logN = log2(N);

ll f[N][logN],log2[N];

//int main

//预处理

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

scanf(“%lld”,&a[i]);

for (ll i = 1; i <= logN; i++)

for (ll j = 1; j + (1 << i) - 1 <= n; j++)

f[j][i] = max(f[j][i - 1], f[j + (1 << (i - 1))][i - 1]);

for (ll i = 2; i <= n; i++)

log2[i] = log2[i / 2] + 1;

//查询

for (ll i = 0; i < m; ++i) {

ll l,r;

scanf(“%lld%lld”,&l,&r);

ll s = log2[r - l + 1];

ll ans=max(f[l][s], f[r - (1 << s) + 1][s]);

}

//更新

template <size\_t N, class T = int>

struct ST {

vector<array<T, \_\_log2(N) + 1>> f;

function<T(T, T)> op;

ST(auto bg, auto ed, function<T(T, T)> op = [](T x, T y)

{return max(x, y); }) : op(op) {

f.resize(N + 1);

size\_t i = 0;

for (auto it = bg; it != ed; ++it)

f[++i][0] = \*it;

for (size\_t i = 1; i <= \_\_lg(N); ++i)

for (size\_t j = 1; j + (1 << i) - 1 <= N; ++j)

f[j][i]=op(f[j][i-1],f[j+(1<<(i-1))][i-1]);

}

T query(size\_t l, size\_t r) {

size\_t s = \_\_lg(r - l + 1);

return op(f[l][s], f[r - (1 << s) + 1][s]);

}

};

## MEX

inline int mex(auto v) { //v可以是vector、set等容器

unordered\_set<int> S;

for (auto e : v)

S.insert(e);

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

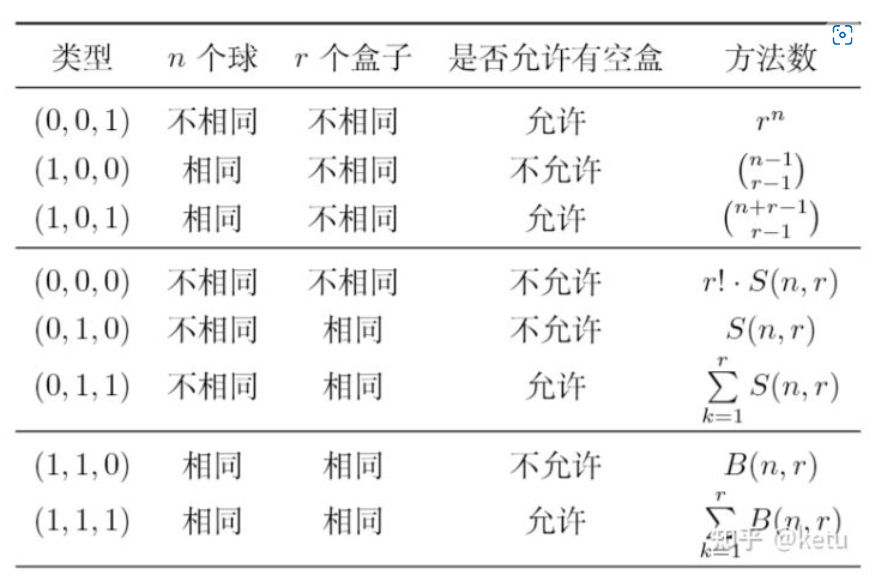
if (S.find(i) == S.end())

return i;

}

## 球盒模型

//(球,盒,空盒) 1为相同,0为不同



## 矩阵快速幂

struct Matrix {

int a[M][M];

int row, col;

inline void Init(int key) {

row = col = key;

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

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

if (i == j) a[i][j] = 1;

else a[i][j] = 0;

}

}

inline Matrix operator + (const Matrix& b) {

Matrix c;

c.row = row;

c.col = col;

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

for (int j = 0; j < col; j++)

c.a[i][j] = (a[i][j] + b.a[i][j]) % mod;

return c;

}

inline Matrix operator + (int x) {

Matrix p = \*this;

for (int i = 0; i < min(row, col); i++)

p.a[i][i] = (p.a[i][i] + x) % mod;

return p;

}

inline Matrix operator - (const Matrix& b) {

Matrix c;

c.row = row;

c.col = col;

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

for (int j = 0; j < col; j++)

c.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;

return c;

}

inline Matrix operator \* (const Matrix& b) {

Matrix c;

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

for (int j = 0; j < b.col; j++) {

c.a[i][j] = 0;

for (int k = 0; k < col; k++)

c.a[i][j] = (c.a[i][j] + 1ll \* a[i][k] \* b.a[k][j] % mod + mod) % mod;

}

c.row = row;

c.col = b.col;

return c;

}

inline Matrix Pow(int t) {

Matrix ans, p = \*this;

ans.Init(p.col);

while (t) {

if (t & 1) ans = ans \* p;

p = p \* p;

t >>= 1;

}

return ans;

}

inline void Print() {

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

for (int j = 0; j < col; j++)

printf("%lld ", a[i][j]);

pn;

}

}

};

## 哈希

ll Hash[hashCnt][N];

ll pwMod[hashCnt][N];

char s[N];

int ls;

inline void init() {

ls = 0;

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

Hash[i][0] = 0;

pwMod[i][0] = 1;

}

}

inline void extend(char c) {

s[++ls] = c;

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

pwMod[i][ls] = pwMod[i][ls - 1] \* hashNum[i] % hashMod[i];

Hash[i][ls] = (Hash[i][ls - 1] \* hashNum[i] + c) % hashMod[i];

}

}

inline vector<ll> getHash(int l, int r) {

vector<ll> res(hashCnt, 0);

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

ll t = (Hash[i][r] - Hash[i][l - 1] \* pwMod[i][r - l + 1]) % hashMod[i];

t = (t + hashMod[i]) % hashMod[i];

res[i] = t;

}

return res;

}

## 字符串匹配

inline void getNext(int next[], string t) {

int j = 0, k = -1;

next[0] = -1;

while (j < t.length() - 1) {

if (k == -1 || t[j] == t[k]) {

j++; k++;

if (t[j] == t[k]) next[j] = next[k];

else next[j] = k;

}

else k = next[k];

}

}

inline int kmp(string s, string t) {

int next[N], i = 0, j = 0;

getNext(next, t);

while (i < s.size() && j < t.size()) {

if (j == -1 || s[i] == t[j]) {

i++;

j++;

}

else j = next[j];

}

if (j >= t.size()) return (i - t.size());

else return (-1);

}

## Dijkstra

ll dis[N];

bool vis[N];

vector<pair<ll, ll> > e[N];

inline void dij(int x) {

memset(dis, 127, sizeof(dis));

memset(vis, false, sizeof(vis));

priority\_queue < pair < ll, ll >, vector < pair < ll, ll > >,greater < pair < ll, ll > > > q;

dis[x] = 0;

q.push(make\_pair(0, x));

while (!q.empty()) {

int u = q.top().second;

q.pop();

if (vis[u]) continue;

vis[u] = true;

for (int i = 0; i < e[u].size(); i++) {

int v = e[u][i].second;

dis[v] = min(dis[v], dis[u] + e[u][i].first);

q.push(make\_pair(dis[v], v));

}

}

}

## 二分图最大匹配

struct augment\_path {

vector<vector<int> > g;

vector<int> pa, pb, vis;

int n, m; // 两个集合的大小

int dfn, res;

augment\_path(int \_n, int \_m) : n(\_n), m(\_m) {

assert(0 <= n && 0 <= m);

pa = vector<int>(n, -1);

pb = vector<int>(m, -1);

vis = vector<int>(n);

g.resize(n);

res = 0;

dfn = 0;

}

inline void add(int from, int to) {

assert(0 <= from && from < n && 0 <= to && to < m);

g[from].push\_back(to);

}

inline bool dfs(int v) {

vis[v] = dfn;

for (int u : g[v])

if (pb[u] == -1) {

pb[u] = v;

pa[v] = u;

return true;

}

for (int u : g[v])

if (vis[pb[u]] != dfn && dfs(pb[u])) {

pa[v] = u;

pb[u] = v;

return true;

}

return false;

}

inline int solve() {

while (true) {

dfn++;

int cnt = 0;

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

if (pa[i] == -1 && dfs(i)) cnt++;

if (cnt == 0) break;

res += cnt;

}

return res;

}

};

#### 最小点覆盖（最大匹配）

//int main

augment\_path e(odd.size(), even.size());

//建两个集合的联系即可，不用内部建边，下标为两个集合各自的下标

int res = e.solve();

#### 最大独立集

//int main

augment\_path e(odd.size(), even.size());

//建两个集合的联系即可，不用内部建边，下标为两个集合各自的下标

int res = n - e.solve();

## 建树

int l;

struct node {

node\* next;

int where;

} \*first[N], a[N];

inline void makeList(int x, int y) { //创建一条从x到y的线

a[++l].where = y;

a[l].next = first[x];

first[x] = &a[l];

}

## 线段树

ll n, q;

ll a[N], sum[5 \* N], mul[5 \* N], laz[5 \* N];

inline void up(int i) {

sum[i] = (sum[(i << 1)] + sum[(i << 1) | 1]) % mod;

}

inline void down(int i, int s, int t) {

int l = (i << 1), r = (i << 1) | 1, mid = (s + t) >> 1;

if (mul[i] != 1) {

mul[l] = mul[l] \* mul[i] % mod;

mul[r] = mul[r] \* mul[i] % mod;

laz[l] = laz[l] \* mul[i] % mod;

laz[r] = laz[r] \* mul[i] % mod;

sum[l] = sum[l] \* mul[i] % mod;

sum[r] = sum[r] \* mul[i] % mod;

mul[i] = 1;

}

if (laz[i]) {

sum[l] = (sum[l] + laz[i] \* (mid - s + 1)) % mod;

sum[r] = (sum[r] + laz[i] \* (t - mid)) % mod;

laz[l] = (laz[l] + laz[i]) % mod;

laz[r] = (laz[r] + laz[i]) % mod;

laz[i] = 0;

}

return;

}

inline void build(int s, int t, int i) {

mul[i] = 1;

if (s == t) {

sum[i] = a[s];

return;

}

int mid = s + ((t - s) >> 1);

build(s, mid, i << 1);

build(mid + 1, t, (i << 1) | 1);

up(i);

}

inline void getMul(int l, int r, int s, int t, int i, ll z) {

int mid = s + ((t - s) >> 1);

if (l <= s && t <= r) {

mul[i] = mul[i] \* z % mod;

laz[i] = laz[i] \* z % mod;

sum[i] = sum[i] \* z % mod;

return;

}

down(i, s, t);

if (mid >= l) getMul(l, r, s, mid, (i << 1), z);

if (mid + 1 <= r) getMul(l, r, mid + 1, t, (i << 1) | 1, z);

up(i);

}

inline void getAdd(int l, int r, int s, int t, int i, ll z) {

int mid = s + ((t - s) >> 1);

if (l <= s && t <= r) {

sum[i] = (sum[i] + z \* (t - s + 1)) % mod;

laz[i] = (laz[i] + z) % mod;

return;

}

down(i, s, t);

if (mid >= l) getAdd(l, r, s, mid, (i << 1), z);

if (mid + 1 <= r) getAdd(l, r, mid + 1, t, (i << 1) | 1, z);

up(i);

}

inline ll getAns(int l, int r, int s, int t, int i) {

int mid = s + ((t - s) >> 1);

ll tot = 0;

if (l <= s && t <= r) return sum[i];

down(i, s, t);

if (mid >= l) tot = (tot + getAns(l, r, s, mid, (i << 1))) % mod;

if (mid + 1 <= r) tot = (tot + getAns(l, r, mid + 1, t, (i << 1) | 1)) % mod;

return tot;

}

//int main

build(1, n, 1);

while (q--) {

int bh = read<int>();

if (bh == 1) {

int x = read<int>(), y = read<int>(), z = read<int>();

getMul(x, y, 1, n, 1, z);

}

else if (bh == 2) {

int x = read<int>(), y = read<int>(), z = read<int>();

getAdd(x, y, 1, n, 1, z);

}

else if (bh == 3) {

int x = read<int>(), y = read<int>();

ll ans = getAns(x, y, 1, n, 1);

write<ll>(ans);

pn;

}

}

#### 区间修改

ll n, q;

ll a[N], b[N], lazy[5 \* N];

inline void push(int xb) {

b[xb] = b[xb \* 2] + b[xb \* 2 + 1];

}

inline void down(int l, int r, int xb) {

if (lazy[xb]) {

int len = r - l + 1;

int lenr = len / 2;

int lenl = len - lenr;

lazy[xb \* 2 + 1] = lazy[xb];

lazy[xb \* 2] = lazy[xb];

b[xb \* 2] = lenl \* lazy[xb];

b[xb \* 2 + 1] = lenr \* lazy[xb];

lazy[xb] = 0;

}

}

inline void build(int l, int r, int xb) {

lazy[xb] = 0;

if (l == r) {

b[xb] = a[l];

return;

}

build(l, (l + r) / 2, xb \* 2);

build((l + r) / 2 + 1, r, xb \* 2 + 1);

push(xb);

}

inline void change(int ll, int rr, int val, int l, int r, int xb) {

if (ll <= l && rr >= r) {

b[xb] = val \* (r - l + 1);

lazy[xb] = val;

return;

}

int m = (l + r) / 2;

down(l, r, xb);

if (ll <= m) change(ll, rr, val, l, m, xb \* 2);

if (rr > m) change(ll, rr, val, m + 1, r, xb \* 2 + 1);

push(xb);

}

inline int query(int ll, int rr, int l, int r, int xb) {

if (ll <= l && rr >= r) return b[xb];

int m = (l + r) / 2;

down(l, r, xb);

int ans = 0;

if (rr > m) ans = ans + query(ll, rr, m + 1, r, xb \* 2 + 1);

if (ll <= m) ans = ans + query(ll, rr, l, m, xb \* 2);

return ans;

}

//int main

build(1, n, 1);

while(q--) {

scanf("%d %d %d", &cmd, &l, &r);

ll ans = 0;

if (cmd) {

scanf("%d", &newp);

change(1, n, l, r, 1, newp);

}

else ans = query(1, n, l, r, 1);

}

## lca

int n, m, from, to, dist, rt;

int in[N], dis[N], fa[N], ans[N];

bool vis[N];

struct node {

int cnt;

int x[N], y[N], z[N], nxt[N], fst[N];

inline void set() {

cnt = 0;

init(x, 0, N);

init(y, 0, N);

init(z, 0, N);

init(nxt, 0, N);

init(fst, 0, N);

}

inline void add(int a, int b, int c) {

x[++cnt] = a;

y[cnt] = b;

z[cnt] = c;

nxt[cnt] = fst[a];

fst[a] = cnt;

}

}e, q;

inline void dfs(int rt) {

for (int i = e.fst[rt]; i; i = e.nxt[i]) {

dis[e.y[i]] = dis[rt] + e.z[i];

dfs(e.y[i]);

}

}

inline int getfa(int k) {

return fa[k] == k ? k : fa[k] = getfa(fa[k]);

}

inline void lca(int rt) {

for (int i = e.fst[rt]; i; i = e.nxt[i]) {

lca(e.y[i]);

fa[getfa(e.y[i])] = rt;

}

vis[rt] = 1;

for (int i = q.fst[rt]; i; i = q.nxt[i])

if (vis[q.y[i]] && !ans[q.z[i]])

ans[q.z[i]] = dis[q.y[i]] + dis[rt] - 2 \* dis[getfa(q.y[i])];

}

//int main

q.set(), e.set();

memset(in, 0, sizeof in);

memset(vis, 0, sizeof vis);

memset(ans, 0, sizeof ans);

scanf("%d %d", &n, &m);

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

scanf("%d %d %d", &from, &to, &dist);

e.add(from, to, dist);

in[to]++;

}

for (int i = 1; i <= m; i++) {

scanf("%d %d", &from, &to);

q.add(from, to, i);

q.add(to, from, i);

}

rt = 0;

for (int i = 1; i <= n && rt == 0; i++)

if (in[i] == 0) rt = i;

dis[rt] = 0;

dfs(rt);

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

fa[i] = i;

lca(rt);

## 最小生成树

int n;

ll ans;

int fa[N];

struct node {

int x, y, z;

inline bool operator < (const node a)const {

return z < a.z;

}

} G[N];

inline int find(int x) {

if (fa[x] == x) return x;

else return fa[x] = find(fa[x]);

}

inline void Kruskal() {

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

fa[i] = i;

sort(G + 1, G + n + 1);

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

int x = find(G[i].x), y = find(G[i].y);

if (x != y) {

fa[find(x)] = find(y);

ans += G[i].z;

}

}

}

//int main

n = read<int>();

ans = 0;

Init<int>(fa, 0, n);

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

int x = read<int>();

int y = read<int>();

int z = read<int>();

G[i] = { x,y,z };

}

Kruskal();

## 字典树

int n, q, tot;

char s[N], word[N];

struct node {

int pass;

int end;

int next;

}trie[N][26];

inline void insert(char s[]) {

int now = 0;

int len = strlen(s);

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

int ch = s[i] - 'a';

if (!trie[now][ch].next) trie[now][ch].next = ++tot;

trie[now][ch].pass++;

if (i == len - 1) trie[now][ch].end++;

else now = trie[now][ch].next;

}

}

inline bool search(char word[]) {

int res = 0;

int now = 0;

int len = strlen(word);

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

int ch = word[i] - 'a';

if (!trie[now][ch].next) return false;

if (i != len - 1) now = trie[now][ch].next;

else res = ch;

}

return trie[now][res].end ? true : false;

}

inline int searchPrefix(char word[]) {

int res = 0;

int now = 0;

int len = strlen(word);

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

int ch = word[i] - 'a';

if (!trie[now][ch].next) break;

if (i != len - 1) now = trie[now][ch].next;

else res = trie[now][ch].pass;

}

return res;

}

inline void del(char word[]) {

if (!search(word)) return;

int now = 0;

int len = strlen(word);

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

int ch = word[i] - 'a';

trie[now][ch].pass--;

if (i != len - 1) now = trie[now][ch].next;

else trie[now][ch].end--;

}

}

//int main

n = read<int>();

q = read<int>();

while (n--) {

scanf("%s", s);

insert(s);

}

while (q--) {

scanf("%s", word);

printf(search(word) ? "YES\n" : "NO\n");

int res = searchPrefix(word);

}

## 德州扑克

string suits = "CDHS"; //花色

string ranks = "23456789TJQKA"; //牌序大小

inline int str2int(string& s) {

int res = 0;

for (int i = 0; i < 4; i++)

if (s[0] == suits[i]) res += i \* 13;

for (int i = 0; i < 13; i++)

if (s[1] == ranks[i]) res += i;

return res;

}

inline vector<int> highest\_hand(vector<string>& aa) {

vector<int> a;

for (auto x : aa)

a.push\_back(str2int(x));

vector<vector<bool> > c(4, vector<bool>(13, false));

vector<int> suits(4, 0);

vector<int> ranks(13, 0);

for (int i = 0; i < (int)a.size(); i++) {

int suit = a[i] / 13;

int rank = a[i] % 13;

c[suit][rank] = true;

suits[suit]++;

ranks[rank]++;

}

vector<vector<int>> n2r(5);

for (int i = 12; i >= 0; i--)

n2r[ranks[i]].push\_back(i);

// 皇家同花顺 Royal flush

for (int i = 0; i < 4; i++)

for (int j = 12; j >= 3; j--) {

bool ok = true;

for (int k = 0; k < 5; k++) {

int rank = (j - k >= 0) ? j - k : 12;

if (!c[i][rank]) {

ok = false;

break;

}

}

if (ok) return { 8, j };

}

// 四条 Four of a kind

if (!n2r[4].empty())

for (int i = 12; i >= 0; i--) {

if (!ranks[i] || i == n2r[4][0]) continue;

return { 7, n2r[4][0], i };

}

// 葫芦 Full house

if (!n2r[3].empty() && (n2r[3].size() + n2r[2].size()) >= 2u)

for (int i = 12; i >= 0; i--) {

if (i == n2r[3][0] || ranks[i] < 2) continue;

return { 6, n2r[3][0], i };

}

// 同花顺 Straight flush

for (int i = 0; i < 4; i++) {

if (suits[i] < 5) continue;

vector<int> res{ 5 };

for (int j = 12; j >= 0; j--) {

if (c[i][j]) res.push\_back(j);

if ((int)res.size() == 6) return res;

}

}

// 顺子 Straight

for (int i = 12; i >= 3; i--) {

bool ok = true;

for (int j = 0; j < 5; j++) {

int rank = (i - j >= 0) ? i - j : 12;

if (!ranks[rank]) {

ok = false;

break;

}

}

if (ok) return { 4, i };

}

// 三条 Three of a kind

if (!n2r[3].empty()) return { 3, n2r[3][0], n2r[1][0], n2r[1][1] };

// 两对 Two pairs

if ((int)n2r[2].size() >= 2) {

int rem = (n2r[2].size() > 2u) ? max(n2r[2][2], n2r[1][0]) : n2r[1][0];

return { 2, n2r[2][0], n2r[2][1], rem };

}

// 一对 Pair

if ((int)n2r[2].size() == 1) return { 1, n2r[2][0], n2r[1][0], n2r[1][1], n2r[1][2] };

// 其他: 高牌 Highcard, 同花 Highcard

vector<int> res{ 0 };

res.insert(res.end(), n2r[1].begin(), n2r[1].begin() + 5);

return res;

}

vector<string> AA, BB, SS;

bool seen[1 << 6][1 << 6];

int win[1 << 6][1 << 6];

vector<string> card;

inline int calc(int rem, int alice, int turn) {

if (seen[rem][alice]) return win[rem][alice];

seen[rem][alice] = true;

if (!rem) {

vector<string> S = AA, T = BB;

for (int i = 0; i < 6; i++) {

if (alice & (1 << i)) S.push\_back(SS[i]);

else T.push\_back(SS[i]);

}

vector<int> ma1 = highest\_hand(S), ma2 = highest\_hand(T);

if (ma1 > ma2) return win[rem][alice] = 1;

else if (ma1 == ma2) return win[rem][alice] = 0;

return win[rem][alice] = -1;

}

else {

if (turn == 0) {

int res = -3;

for (int i = 0; i < 6; i++) {

if (!(rem & (1 << i))) continue;

res = max(res, calc(rem - (1 << i), alice | (1 << i), turn ^ 1));

if (res == 1) break;

}

return win[rem][alice] = res;

}

else {

int res = 3;

for (int i = 0; i < 6; i++) {

if (!(rem & (1 << i))) continue;

res = min(res, calc(rem - (1 << i), alice, turn ^ 1));

if (res == -1) break;

}

return win[rem][alice] = res;

}

}

}

// AA,BB中先花色，再牌序

## 斗地主

#### 双王算对子

int n, ans;

int pai[20]; // 1是王, 14是A

inline int sanPai();

inline void feiJi(int step);

inline void shunZi(int step);

inline void lianDui(int step);

inline void chuPai(int step) { //出牌

if (step >= ans) return;

int tmp = sanPai(); //打散牌

ans = min(tmp + step, ans);

feiJi(step); //飞机

shunZi(step); //顺子

lianDui(step); //连对

}

inline void feiJi(int step) { //飞机

for (int st = 3; st <= 13; st++) {

int l = 0;

while (pai[st + l] >= 3)

l++;

for (int j = l; j >= 2; j--) {

int end = st + j - 1;

for (int k = st; k <= end; k++)

pai[k] -= 3; //飞机

chuPai(step + 1);

for (int k = st; k <= end; k++)

pai[k] += 3;

}

}

}

inline void lianDui(int step) { //连对

for (int st = 3; st <= 12; st++) {

int l = 0;

while (pai[st + l] >= 2)

l++;

for (int j = l; j >= 3; j--) {

int end = st + j - 1;

for (int k = st; k <= end; k++)

pai[k] -= 2; //连对

chuPai(step + 1);

for (int k = st; k <= end; k++)

pai[k] += 2;

}

}

}

inline void shunZi(int step) { //顺子

for (int st = 3; st <= 10; st++) {

int l = 0;

while (pai[st + l] >= 1)

l++;

for (int j = l; j >= 5; j--) {

int end = st + j - 1;

for (int k = st; k <= end; k++)

pai[k]--; //顺子

chuPai(step + 1);

for (int k = st; k <= end; k++)

pai[k]++;

}

}

}

inline int sanPai() { //贪心打散牌

int num = 0;

int zs[5];

memset(zs, 0, sizeof(zs));

bool wangZha = false; //是否有王炸

if (pai[1] == 2) wangZha = true;

zs[1] += pai[1];

for (int i = 2; i <= 14; i++)

zs[pai[i]]++;

while (!zs[3] && zs[1] == 1 && zs[2] == 1 && zs[4] > 1) {

zs[4] -= 2;

zs[1]--;

zs[2]--;

num += 2;

}

//把一个炸拆成3张和单牌,再出一组四带二单和三带一

while (!zs[2] && zs[1] == 1 && zs[4] == 1 && zs[3] > 1) {

zs[3] -= 2;

zs[1]--;

zs[4]--;

num += 2;

}

//把一组三张拆成一对和一单,再出一组四带二单和三带二

if (zs[3] + zs[4] > zs[1] + zs[2])//三四张的比单牌和对牌多,拆着打

while (zs[4] && zs[2] && zs[3]) {

zs[2]--;

zs[3]--;

zs[1]++;

zs[4]--;

num++;

}

//拆三张,4带两对余一单

if (zs[3] + zs[4] > zs[1] + zs[2])//还多继续拆

while (zs[4] && zs[1] && zs[3]) {

zs[1]--;

zs[3]--;

zs[2]++;

zs[4]--;

num++;

}

//拆三张,4带两单余一对

while (zs[4] && zs[1] > 1) {

zs[4]--;

zs[1] -= 2;

num++;

}

//四带两单

while (zs[4] && zs[2] > 1) {

zs[4]--;

zs[2] -= 2;

num++;

}

//四带两对

while (zs[4] && zs[2]) {

zs[4]--;

zs[2]--;

num++;

}

//对看成两单再四带

if (zs[3] % 3 == 0 && zs[1] + zs[2] <= 1) //三张的太多了拆三张

while (zs[3] > 2) {

zs[3] -= 3;

num += 2;

}

//把一组三张拆成单和对,再出三带一和三带二

while (zs[3] && zs[1]) {

zs[3]--;

zs[1]--;

num++;

}

//三带一

while (zs[3] && zs[2]) {

zs[3]--;

zs[2]--;

num++;

}

//三带二

//还剩三张和炸,组合出

while (zs[4] > 1 && zs[3]) {

zs[3]--;

zs[4] -= 2;

num += 2;

}

//把一个炸拆成一对和两单,再出三带二和四带两单

while (zs[3] > 1 && zs[4]) {

zs[4]--;

zs[3] -= 2;

num += 2;

}

//把一个炸拆成两对,再出两组三带一对

while (zs[3] > 2) {

zs[3] -= 3;

num += 2;

}

//同上,把一组三张拆成单和对,再出三带一和三带二

while (zs[4] > 1) {

zs[4] -= 2;

num++;

}

//把一个炸拆成两对,再出一组四带两对

if (wangZha && zs[1] >= 2) return num + zs[2] + zs[3] + zs[4] + zs[1] - 1;//双王一块出

else return num + zs[1] + zs[2] + zs[3] + zs[4];//出剩余的牌,返回答案

}

#### 双王不算对子

int n;

struct cardset {

int count[15];

int tot;

cardset() {

for (int i = 0; i <= 14; i++)

count[i] = 0;

tot = 0;

}

inline void add(int number, int c) {

count[number] += c;

tot += c;

}

inline bool operator < (const cardset& s)const {

if (tot < s.tot) return true;

else if (tot > s.tot) return false;

for (int i = 1; i <= 14; i++)

if (count[i] != s.count[i]) return count[i] < s.count[i];

return false;

}

};

//debug

inline ostream& operator << (ostream& stream, const cardset& set) {

for (int i = 1; i <= 14; i++)

for (int k = 0; k < set.count[i]; k++)

stream << i << ",";

return stream;

}

struct state {

int count[15]; // 1:Joker 14:Ace

int tot;

int upperbound;

int step;

cardset lastplay;

state(int n) {

for (int i = 0; i < 15; i++)

count[i] = 0;

tot = upperbound = n;

step = 0;

lastplay.tot = 0;

}

state(const state& s) {

\*this = s;

}

inline void play(const cardset& s) {

tot -= s.tot;

step++;

for (int i = 1; i <= 14; i++)

count[i] -= s.count[i];

upperbound = s.tot;

lastplay = s;

}

};

inline void find(const state& s, vector<cardset>& plays, int lowerbound, int upperbound) {

if (upperbound < lowerbound) return;

// 顺子

for (int i = 3; i <= 13; i++)

if (s.count[i] >= 3) {

cardset play;

play.add(i, 3);

int seq = 1;

for (int j = i + 1; j <= 14; j++) {

if (s.count[j] < 3) break;

else {

seq++;

play.add(j, 3);

if (seq >= 2) {

if (play.tot >= lowerbound && play.tot <= upperbound)

plays.push\_back(play);

}

}

}

}

for (int i = 3; i <= 12; i++)

if (s.count[i] >= 2) {

cardset play;

play.add(i, 2);

int seq = 1;

for (int j = i + 1; j <= 14; j++) {

if (s.count[j] < 2) break;

else {

seq++;

play.add(j, 2);

if (seq >= 3) {

if (play.tot >= lowerbound && play.tot <= upperbound)

plays.push\_back(play);

}

}

}

}

for (int i = 3; i <= 10; i++)

if (s.count[i] >= 1) {

cardset play;

play.add(i, 1);

int seq = 1;

for (int j = i + 1; j <= 14; j++) {

if (s.count[j] < 1) break;

else {

seq++;

play.add(j, 1);

if (seq >= 5) {

if (play.tot >= lowerbound && play.tot <= upperbound)

plays.push\_back(play);

}

}

}

}

// 火箭

if (s.count[1] == 2 && lowerbound <= 2 && upperbound >= 2) {

cardset rocket;

rocket.add(1, 2);

plays.push\_back(rocket);

}

// 炸弹

if (lowerbound <= 4 && upperbound >= 4)

for (int i = 1; i <= 14; i++) {

cardset boom;

if (s.count[i] == 4) boom.add(i, 4);

}

// 三带二

if (lowerbound <= 5 && upperbound >= 5)

for (int i = 2; i <= 14; i++)

if (s.count[i] >= 3) { //exclude joke

cardset three;

three.add(i, 3);

for (int j = 2; j <= 14; j++)

if (j != i && s.count[j] >= 2) {

cardset three2 = three;

three2.add(j, 2);

plays.push\_back(three2);

}

}

//三带一

if (lowerbound <= 4 && upperbound >= 4)

for (int i = 2; i <= 14; i++)

if (s.count[i] >= 3) { //exclude joke

cardset three;

three.add(i, 3);

for (int j = 1; j <= 14; j++)

if (j != i && s.count[j] >= 1) {

cardset three1 = three;

three1.add(j, 1);

plays.push\_back(three1);

}

}

// 三张

if (lowerbound <= 3 && upperbound >= 3)

for (int i = 2; i <= 14; i++)

if (s.count[i] >= 3) { //exclude joke

cardset three;

three.add(i, 3);

plays.push\_back(three);

}

// 对子

if (lowerbound <= 2 && upperbound >= 2)

for (int i = 2; i <= 14; i++)

if (s.count[i] >= 2) { //exclude joke

cardset couple;

couple.add(i, 2);

plays.push\_back(couple);

}

// 单牌

if (lowerbound <= 1 && upperbound >= 1)

for (int i = 1; i <= 14; i++)

if (s.count[i] >= 1) { //include joke

cardset single;

single.add(i, 1);

plays.push\_back(single);

}

// 4+2 : 4+4, 4+1couple, 4+2couple, 4+2single

if (lowerbound <= 8 && upperbound >= 4) {

for (int i = 2; i <= 14; i++)

if (s.count[i] >= 4) {

cardset four;

four.add(i, 4);

// 4+4

if (lowerbound <= 8 && upperbound >= 8)

for (int j = 2; j <= 14; j++)

if (j != i && s.count[j] >= 4) {

cardset fourfour = four;

fourfour.add(j, 4);

plays.push\_back(fourfour);

}

// 4+1couple

for (int j = 2; j <= 14; j++)

if (j != i && s.count[j] >= 2) { //Joker excluded

cardset four1couple = four;

four1couple.add(j, 2);

if (lowerbound <= 6 && upperbound >= 6)

plays.push\_back(four1couple);

// 4+2couple

if (lowerbound <= 8 && upperbound >= 8)

for (int k = j + 1; k <= 14; k++)

if (k != i && s.count[k] >= 2) {

cardset four2couple = four1couple;

four2couple.add(k, 2);

plays.push\_back(four2couple);

}

}

// 4+Jokers

if (s.count[1] == 2 && lowerbound <= 6 && upperbound >= 6) {

cardset four2joker = four;

four2joker.add(1, 2);

plays.push\_back(four2joker);

}

// 4+2single

if (lowerbound <= 6 && upperbound >= 6)

for (int j = 1; j <= 14; j++)

if (j != i && s.count[j] >= 1) {

for (int k = j + 1; k <= 14; k++)

if (k != i && s.count[k] >= 1) {

cardset four2single = four;

four2single.add(j, 1);

four2single.add(k, 1);

plays.push\_back(four2single);

}

}

}

}

}

inline int idastar(const state& s, int limit) {

if (!s.tot) return s.step;

int heuristic = (s.tot - 1) / s.upperbound + 1;

if (s.step + heuristic > limit) return 0;

int lowerbound = (s.tot - 1) / (limit - s.step) + 1;

vector<cardset> plays;

find(s, plays, lowerbound, s.upperbound);

for (int i = 0; i < plays.size(); i++) {

state next(s);

if (s.lastplay.tot == 0 || !(s.lastplay < plays[i])) {

next.play(plays[i]);

int finish = idastar(next, limit);

if (finish) return finish;

}

}

return 0;

}

//int main

int res = 0;

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

res = idastar(m, i);

if (res > 0) break;

}

## 对数器

inline void rightCode(vector<ll>& nums) {

}

inline void testCode(vector<ll>& nums) {

}

inline vector<ll> generateRandomVector(int size, int value) {

srand((ll)time(NULL));

vector<ll> result(rand() % (size + 1));

for (ll i = 0; i < result.size(); i++)

result[i] = rand() % (value + 1);

return result;

}

inline void test(ll test\_time, ll size, ll value) {

//test\_time 测试次数，设置比较大，排除特殊情况

//size 生成数组最大尺寸

//value 生成数组每个元素的最大值

bool if\_accept = true;

for (ll i = 0; i < test\_time; i++) {

vector<ll> nums(generateRandomVector(size, value));

vector<ll> nums1(nums);

vector<ll> nums2(nums);

rightCode(nums1);

testCode(nums2); //时间复杂度为算法的时间复杂度乘size

if (nums1 != nums2) {

if\_accept = false;

for (auto c : nums)

printf("%lld ", c);

break;

}

}

printf(if\_accept ? "true!\n" : "false!\n");

}

//int main   
ll test\_time, size, value;  
scanf("%lld %lld %lld", &test\_time, &size, &value);  
test(test\_time, size, value);