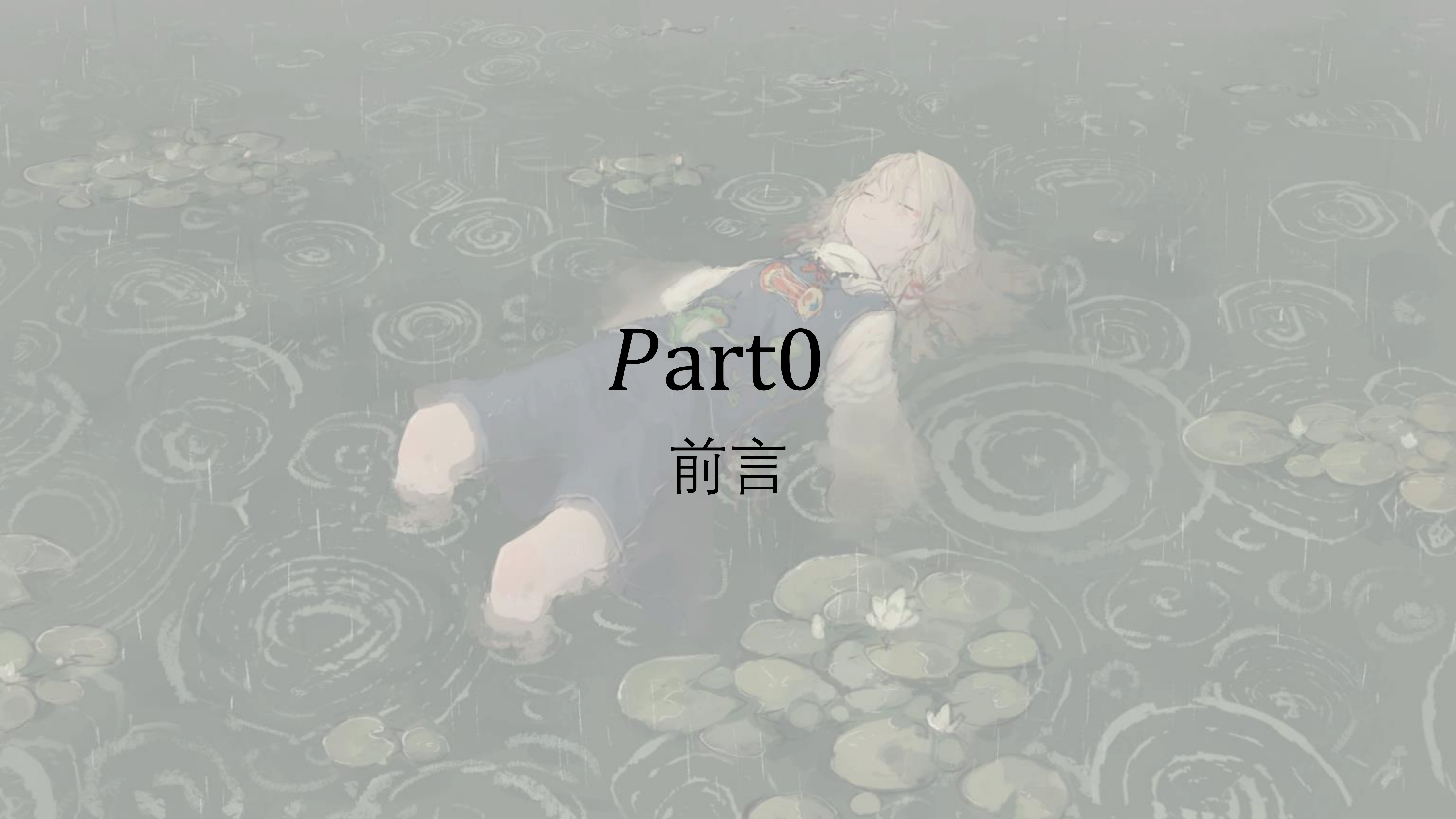
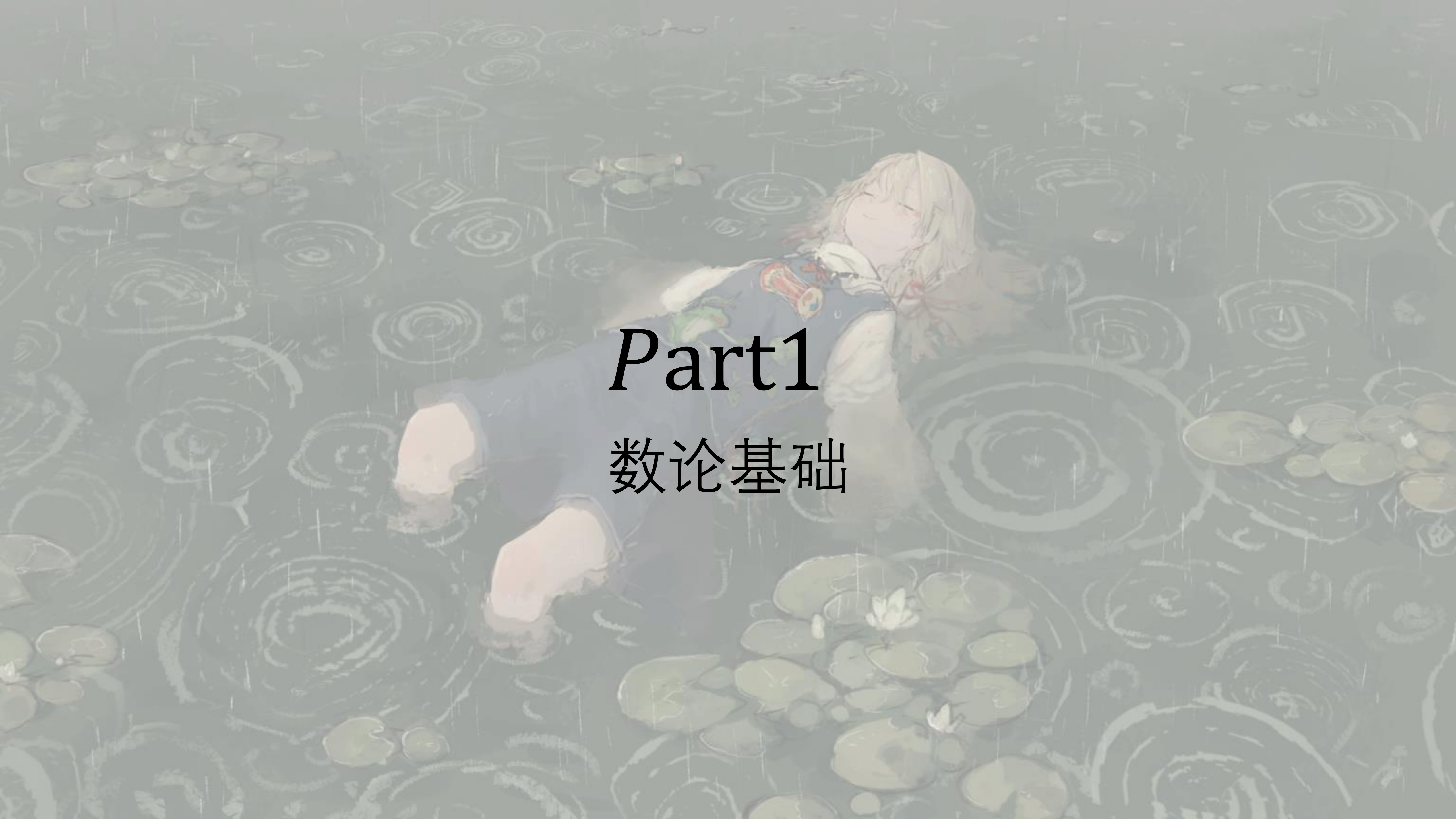


Part0

前言

ACM中的数学

- 数论
- 组合数学
- 概率期望
- 计算几何
- 线性代数
- 多项式
- 博弈论



Part1

数论基础

基本概念 整除 素数 因子 公因子 互素

- 整除: $a \mid b$

  互素: $(a, b) = 1$

  枚举因子： $O(\sqrt{n})$
- 唯一分解定理： 对任意n，

  $n = p_1^{l_1} p_2^{l_2} * \cdots * p_k^{l_k}$，

  $p_1, \ldots, p_k$为互异素数， $l_i \geq 0$
- 素因数分解：
- 求最大公因数（gcd）：

```
void Fact(int x) {
    k = 0;
    for(int i = 2; i * i <= x; i++)
        if(x % i == 0) {
            p[++k] = i; l[k] = 0;
            while(x % i == 0) {
                x /= i; ++l[k];
            }
        }
    if(x != 1) {  // 此时x一定是素数，
        p[++k] = x; l[k] = 1;
```

```
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
```

基本概念 取模与逆元

- 取模与同余
- a 对 b 取模：取a除以b的余数r $(0 \leq r < b)$
- a, b模m同余：a与b模m的余数相同
  $a \equiv b\ (mod\ m) \Leftrightarrow m \mid (b - a)$
- 模意义下有加法和乘法

$$(a + b)\ \%\ m = (a\ \%\ m + b\ \%\ m)\ \%\ m$$

$$(a\ *\ b)\ \%\ m = (a\%m) * (b\%m)\ \%\ m$$

写代码时

- **注1**：C++中$a\%b$有可能为负数（比如$a<0,b>0$时），因此取模后答案一定要输出为$(ans\ +\ mod)\ \%\ mod$
- **注2**：小心溢出

```
const int mod = 1e9 + 7;
int x, y;
int z = x * y % mod; //溢出
int w = 1ll * x * y % mod; //可以
ll x, y;
ll ans = x * y % mod;    //可以
```

- **注3**：对2^64取模，用unsigned long long 自然溢出即可

基本概念 模意义逆元

- 实际上一般情况下，题目会限定一个固定的模数P，取模意义下的运算其实在一个新的运算空间下进行的，其值域为$[0, P-1]$
- 加减乘都好处理，那么除法呢？如何算$b/a\%p$？(a整除b)
- 设$b = a * m$，考虑，如果能够找到一个x，使得

$$ax \equiv 1 \ (mod\ p)$$

那么就有，$b * x \equiv m \ (mod\ p)$，就求到了$b/a\%p$的值

- 我们称x为a在模p意义下的逆元，问题转化为如何求逆元。
- 在求逆元之前先别太着急，因为并不是对所有的a和p都有逆元可求，事实上，**逆元的存在条件是 $\boldsymbol{a, p}$ 互素**

求逆元

- 有三种方法，分别适用于不同情形，建议都掌握
- https://oi-wiki.org/math/inverse/
- 这里只介绍一种最常用的方法，对于今天的题目来说足够

求逆元 费马小定理 + 快速幂

- 费马小定理:
- 若p为素数且$(a, p)\ =\ 1$，则$a^{p-1} \equiv 1\ (mod\ p)$
- 于是a^{p-2}就是a模p的逆元，快速幂即可
- **注1**：模为素数（今天题目里模都是素数）
- **注2**: $\boldsymbol{a}$如果被$\boldsymbol{p}$整除，则不能求逆元。那么算b/a取模时就要采取其他方法。
- 欧拉定理：费马小定理的推广
- 若$(a, n)\ =\ 1$(n可以不是素数），则$a^{\varphi(n)} \equiv 1\ (mod\ n)$
- 欧拉函数$\varphi(\mathrm{n}) = |\{1 \le x \le n | (x, n) = 1\}|$，$\varphi(\mathrm{p}) = p - 1$($p$素)

无法求逆元的特殊情况

现在你要算一个分式，取模，它的分母可以整除分子，但分母没有逆元，分子又太大了，超过long long 范围。

如果你先把分子取模，再直接除掉分母，显然是不行的。

一些解决方案：

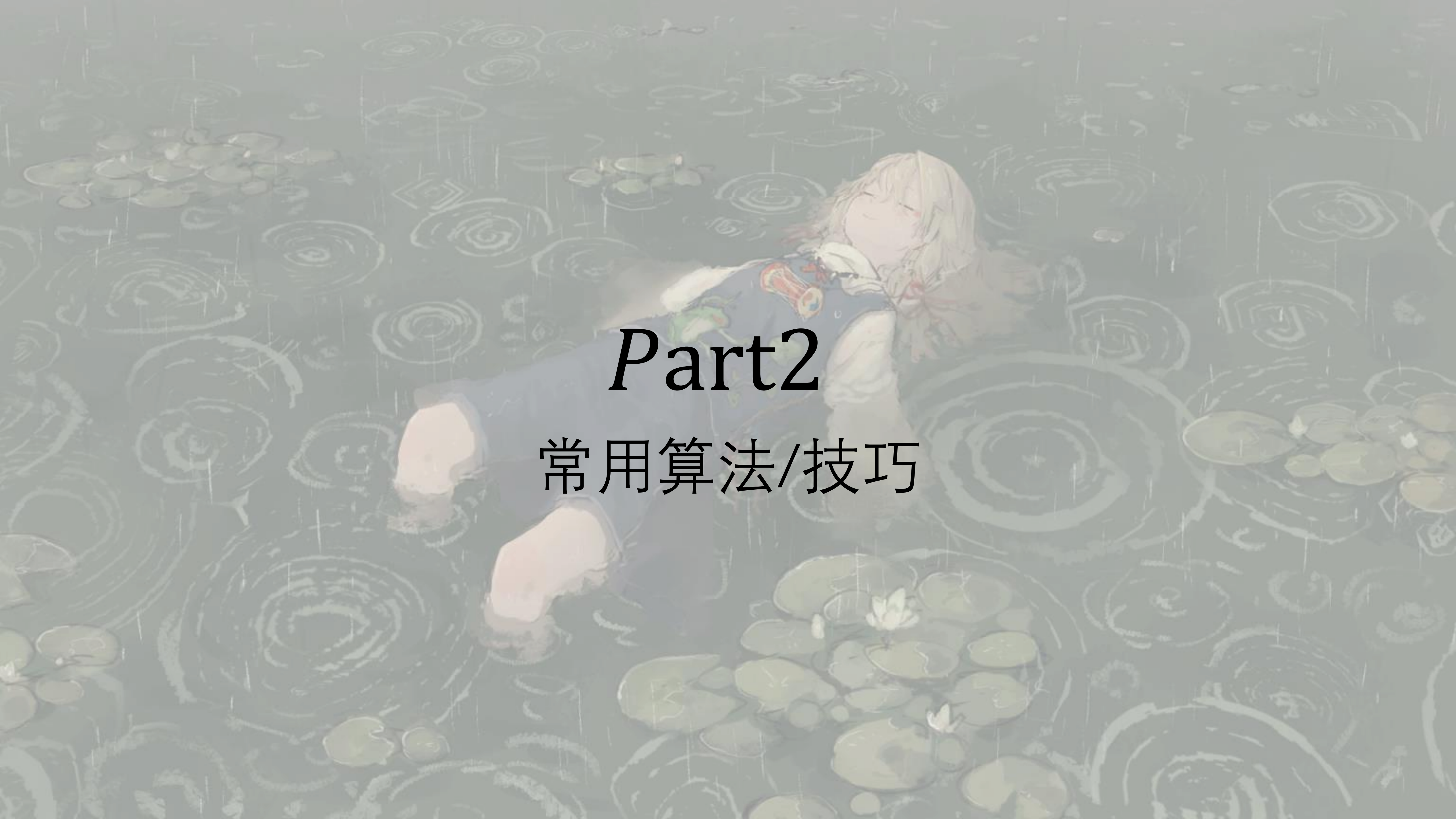
- 当整个分式就很简单（比如n(n+1)/2），直接分类讨论，这里对n讨论奇偶性即可
- 当b比较小，比如$a/2$，$a/6$，可采用

$$(a\ /\ b)\ \%\ p\ =\ (a\ \%\ (b * p))\ /\ b\ \%\ p;$$

~~（证明很简单）~~

有理数取模

- 推广来说，即便a/b不为整数，我们也可以对它取模，也就是有理数取模
- $a/b \ mod \ p \ = \ a * b^{-1} \ mod \ p$
- 合理性：先加/乘再取模 = 先取模再加/乘
- $\left(\frac{a}{b} * \frac{c}{d}\right) mod\ p = \frac{\mathrm{ac}}{\mathrm{bd}} \mathrm{mod\ p} = (\mathrm{a} * \mathrm{b}^{-1}) * (\mathrm{c} * \mathrm{d}^{-1}) \mathrm{\ mod\ p}$ $= \left(\frac{a}{b}\ mod\ p\right) * \left(\frac{c}{d}\ mod\ p\right) mod\ p$
- $\left(\frac{a}{b} + \frac{c}{d}\right) mod\ p = \frac{ad+bc}{bd} mod\ p = (\mathrm{a} * \mathrm{b}^{-1}) + (\mathrm{c} * \mathrm{d}^{-1}) \mathrm{\ mod\ p}$ $= \left(\frac{a}{b}\ mod\ p\right) + \left(\frac{c}{d}\ mod\ p\right) mod\ p$
- 有些题目答案就是有理数，一般会要求以$a * b^{-1}$形式输出
  这时你只需要把所有“除号”换成“乘逆元”就行了。



Part2

常用算法/技巧

常用公式

- $1+2+\cdots+n=\frac{\mathrm{n}(n+1)}{2}$
- $1^2+2^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6}$
- $1^3+2^3+\cdots+n^3=\left[\frac{\mathrm{n}(n+1)}{2}\right]^2$
- $1^k+2^k+\cdots+n^k=?$
- 拉格朗日插值：对单组n,k，$O(\mathrm{k})$求解答案
- http://aequa.me/index.php/2018/02/01/powersum-linear/
- https://codeforces.com/problemset/problem/622/F
- （Lagrange插值属于比较进阶的内容，有兴趣可作了解）

前缀和 虽然它无处不在

- 拉格朗日插值里有个比较简单的子问题:
- n, m给定，要求对 $i: 0\ to\ m$ 求出
- $n*(n-1)*\cdots*(n-i+1)*(n-i-1)*\cdots*(n-m)$
- 先求出n到n – m的累乘，然后对每个i除掉n – i？
- 不妨将它分成前后两部分，先预处理出所有前缀后缀积，然后两两乘在一起即可。

快速幂

- 如何快速地求a^k mod m $(k \geq 0)$
- 如果k是2的幂，那把就a平方，再平方……
- 对一般情况，相当于

在k的二进制位上递推：

- $k = 2 * m + r , r = 0\ or\ 1$
- $a^k = \left(a^2\right)^m * a^r$

```
ll Pow(ll a, ll k, ll p) {
    ll ans = 1;
    while(k) {
        if(k & 1) ans = ans * a % p;
        k >>= 1; a = a * a % p;
    }
    return ans;
}
```

再谈快速幂

- 幂本质上是一个元素 a 自乘 k 次
- a可以是整数，矩阵，多项式，序列 / 函数
- “乘”也可以是加法，矩阵乘，多项式乘积，狄利克雷卷积……
- 比如当题目中模数超过int表示范围（如1e18）（一般不会）
  即便 a, b 已经取模，$a * b$ 仍会直接爆long long，
  考虑乘法就是把a自加b次，而这里两数相加不会溢出，
  把快速幂中的乘法换成加法即可
- 俗称“慢速乘”

矩阵快速幂

- 经典问题：求Fibonacci数列第10亿项 mod m
- $a_{n+1} = a_n + a_{n-1}$，一个递推法则 重复 k 次
- 用状态的转移来理解。
- 矩阵：刻画 向量的变换，状态的转移
- $\begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \end{pmatrix}$，作k次递推 = 左乘 A^{k} → 快速幂
- 矩阵乘法按照$O(n^3)$写即可
- 对于更复杂的递推式（如$a_{n+1} = a_n + 2a_{n-1} + \mathrm{n}^4 + 3^{\mathrm{n}}$）
- 构建状态：在递推的过程中，新状态各项应为原状态的线性组合

矩阵快速幂

- $a_{n+1} = a_n + 2a_{n-1} + \mathrm{n}^4 + 3^{\mathrm{n}}$

$$\begin{pmatrix} a_{n+1} \\ a_n \\ (\mathrm{n}+1)^4 \\ 3^{\mathrm{n}+1} \end{pmatrix} \leftarrow \begin{pmatrix} a_n \\ a_{n-1} \\ \mathrm{n}^4 \\ 3^{\mathrm{n}} \end{pmatrix}$$ 似乎不太好转移

- 添加更多的状态，使转移变得简单
- 进阶：矩阵快速幂优化动态规划
  例题：……

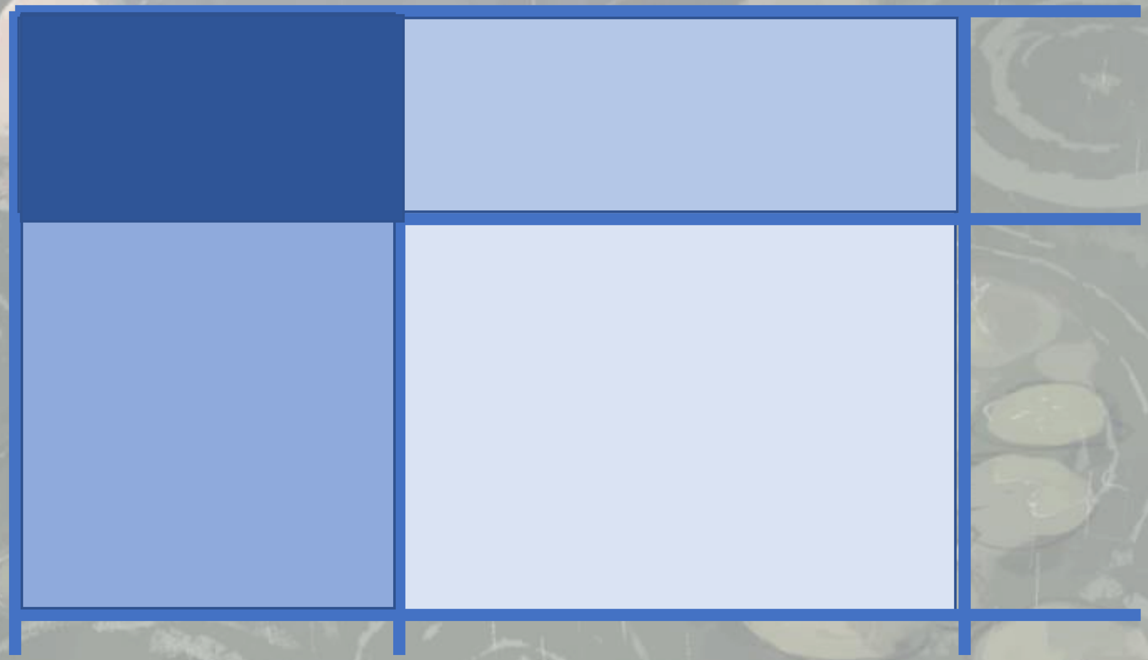
容斥原理

- 设 U 中元素有 n 种不同的属性，而第 i 种属性称为 P_i ，拥有属性 P_i 的元素构成集合 S_i，那么

$$\left|\bigcup_{i=1}^{n} S_i\right| = \sum_i |S_i| - \sum_{i<j} |S_i \cap S_j| + \sum_{i<j<k} |S_i \cap S_j \cap S_k| - \cdots + (-1)^{m-1} \sum_{a_i<a_{i+1}} \left|\bigcap_{i=1}^{m} S_{a_i}\right| + \cdots + (-1)^{n-1} |S_1 \cap \cdots \cap S_n|$$

容斥原理

- 简单的容斥：
- 静态查询序列上一段区间的和：预处理前缀和作差
- 静态查询二维数组上一段方区的和：预处理、二维容斥



容斥原理

- 推导欧拉函数公式： $\varphi(\mathrm{n}) = |\{1 \le x \le n | (x, n) = 1\}|$
- 计数：所有与n互素的数，与n互素：不被n的任意素因子整除
- 容斥模型：S_i = 被素因子p_i整除，取补集

$$\left|\bigcap_i \overline{S_i}\right| = |U| - \left|\bigcup_i S_i\right| = \text{容斥}$$

$$phi(n) = \sum_{A \subseteq \{p1, p2, \ldots, pk\}} (-1)^{|A|} n / \prod_{p \in A} p = n(1 - \frac{1}{p_1}) \ldots (1 - \frac{1}{p_k})$$

容斥原理

- 怎样代码实现？
- 容斥其实是在“枚举子集”：按子集元素个数决定系数为1还是-1，一个子集的贡献值由题目而定
- 枚举子集的方法：二进制啊！
- 比如10个元素的集合，它有1024个子集，每个子集都与一个十位二进制数对应，所以你只要令i从0循环到1023，对每个i利用C++位运算获取出子集包含哪些元素，然后计算贡献即可。
- 一般来说，需要你枚举子集的集合大小（也就是“属性”的数量）顶多20左右，因为2^20 ≈ 1e6（这种数据范围也在暗示你容斥）

抽屉原理

- 你有n+1个苹果，想要放到n个抽屉里，那么必然会有至少一个抽屉里有两个（或以上）的苹果。
- 它是一个存在性定理，看起来和算法没什么关系
- 但是ACM里并不乏一些通过 大胆猜想/证明/构造法 可以直接得到答案的题目，抽屉原理有时确实会用上

素数筛法

- 素数筛法：筛出1~n中的所有素数（1不是素数）
- 思路：i从小到大遍历，筛去i的倍数，用这种方法筛掉所有的合数，剩下的就是素数。
- 埃氏筛：i从2到n，从已获得的素数表中从小到大枚举$p[j]$，筛去$i*p[j]$,(当这个数超过n自然要break)
- 时间复杂度至多为$O(nlgn)$,(其实是$O(nlglgn)$,不会证)
- 这种方法的优化空间：一个合数被重复筛去多次，如果一个合数只被筛去一次，就可以$O(n)$地筛素数了 → 线性筛

线性筛 / 欧拉筛

- x为合数，它可能有多个素因子。设它是被$i * prime[j]$筛掉的，那么满足条件的$prime[j]$就不止一个。
- 只要在筛的过程中保证: $prime[j]$是被筛数的**最小素因子**
- 一旦$prime[j] \mid i$, 再往后$p[j]$就不再作为最小素因子，剩下的合数就不归它筛了

```
void Prime(int n) {
    isnt[1] = true;
    cnt = 0;
    for (int i = 2; i <= n; i++) {
        if (!isnt[i]) prime[++cnt] = i;
        for (int j = 1; j <= cnt; j++) {
            if (1ll * i * prime[j] > n) break;
            isnt[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
```

积性函数

- 一类满足$f(ab) = f(a)f(b)$ $(gcd(a, b) = 1)$的函数
- 若去掉互素的限制条件，则为完全积性函数
- 对于x，将其质因数分解$x = p_1^{q_1}\ p_2^{q_2} * \cdots * p_k^{q_k}$，则
- $f(x) = f\left(p_1^{q_1}\right)f\left(p_2^{q_2}\right) \dots f\left(p_k^{q_k}\right)$
- 也就是说$f(x)$可以分解为若干个因子的乘积，其中每个因子都是一个素数幂的函数值。
- 欧拉函数是积性函数。
- 线性筛积性函数：在之前的线性筛中，我们总是通过一个素数p_j来筛到合数$x = i * p_j$，结合这个分解式，可以发现只要通过讨论p_j在i中的幂次，就可以用$f(i)$来递推得到$f(x)$（不过前提是你弄明白了$f(p^q)$怎么算）

线性筛积性函数

1: i是素数 2: p_j是x的重因子 3: p_j第一次出现

欧拉函数： $\varphi(p^q) = p^{q-1} * (p-1)$

$1.\ phi(i) = i - 1$

$2.\ phi(x) = phi(i) * p_j$

$3.\ phi(x) = phi(i) * (p_j - 1)$

- 对于欧拉函数，只要p_j是x的重因子，递推关系是一致的.
- 而对于一些复杂的积性函数，你可以先把x中所有的p_j因子提出来，得到$x = xx * p_j^q$，此时两者就是互素的了，如果$xx > 1$, 可以直接利用积性递推，如果$xx = 1$, 直接代入$f(p^q)$公式

```
for(int i = 2; i <= n; i++) {
    if(!isnt[i]) {
        prime[++cnt] = i;
        //1   1.(i = p)
    }
    for(j) ← 设x = i * p_j , i = p_1^{q_1} p_2^{q_2} ... p_n^{q_n}
        int x = i * prime[j];
        if(x > n) break;
        isnt[x] = 1;
        if(i % prime[j] == 0) {
            //2   2.x = p_1^{q_1+1} p_2^{q_2} ... p_n^{q_n}
            break;
        } else {
            //3   3.x = p_j p_1^{q_1} p_2^{q_2} ... p_n^{q_n}
        }
    }
}
```

积性函数 性质及运算

- 啥？复杂的积性函数？表达式都那么复杂了，还咋判断积性啊？
- 性质：若$f(x)$和$g(x)$均为积性函数，则以下函数也是积性函数

$$h(x) = f(x^p)$$
$$h(x) = f^p(x)$$
$$h(x) = f(x)g(x)$$
$$h(x) = \sum_{d|x} f(d)g(\frac{x}{d})$$

←狄利克雷卷积

- 运算：Dirichlet卷积：使积性函数组成了一个新的运算空间，在这里，我们可以考虑单位元，逆元……

Dirichlet卷积 体验版

- 对于两个数论函数f,g（可以非积性）
  定义它们的Dirichlet卷积 $h = f * g$ 为一个新的数论函数，满足：
$$h(n) = \sum_{d|\mathrm{n}} f(d)g\left(\frac{n}{d}\right)$$
- 两个积性函数的卷积仍为积性函数

- 卷积与反演是数论的特色，不得不品尝（今天不品尝）
- 解锁更多新内容：https://oi-wiki.org/math/mobius/

Dirichlet卷积 体验版

- $h(n)=\sum_{d|\mathrm{n}} f(d)g\left(\frac{n}{d}\right)$
- 不过，在此我们先考虑一个直接的问题，已知f,g各项，怎样快速求$f*g$?
- 如果f,g均为已知公式的积性函数，固然可以线性筛，难点在于手推式子。
- 所以想寻找更一般的方法
- 直接求的复杂度为$O(N^{\frac{3}{2}})$ 有时不能接受
- 从1到n逐个枚举因子总是会在**无效因子**上付出惨痛代价，把思维逆转过来，**从因子出发去枚举倍数**、计算贡献(也就是固定d,枚举n)，当倍数超过N我们直接跳出，这样就不会造成浪费，而时间复杂度为$O(N\left(1+\frac{1}{2}+\cdots+\frac{1}{N}\right))=O(NlgN)$，这与素数筛法的思路是一样的。

数论分块

- 对于一些卷积/反演题，有时卷着卷着就卷出来一个巨复杂的积性函数，然后你要开始筛
- 或者有时演着演着演出来一个带$\left\lfloor\frac{n}{i}\right\rfloor$的函数，比如 $\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$
- 一般用数论分块来处理这些式子，它是许多数论推式子题的基操。

数论分块

对于公式$\sum_{i=1}^{n} f(i)g(\lfloor \frac{n}{i} \rfloor)$，注意到$\lfloor \frac{n}{i} \rfloor$（$1 \le i \le n$）只有$O(\sqrt{n})$种不同的取值

按照$\lfloor \frac{n}{i} \rfloor$的取值将区间$[1, n]$分为一段段连续的区间，形象地称为“分块”

对于任意$i \le n$，我们想找到最大的$i \le j \le n$，使得$\lfloor \frac{n}{i} \rfloor = \lfloor \frac{n}{j} \rfloor$，事实上$j = \frac{n}{\lfloor \frac{n}{i} \rfloor}$

于是每次以$[i, j]$为一小块，分块求和（把g提出，f前缀和作差）

例：$\sum_{i=1}^{n} i * \lfloor \frac{n}{i} \rfloor, n \le 1e13$

数论分块 代码

求 $\sum_{i=a}^{b} i * \lfloor \frac{n}{i} \rfloor, 1 \le a \le b \le n$

```
for(int i = a, j; i <= b; i = j + 1) {
    j = min(n / (n / i), b);
    ans += (n / i) * (pre(j) - pre(i - 1));
}
```

求 $\sum_{i=1}^{\infty} \lfloor \frac{n}{i} \rfloor \lfloor \frac{m}{i} \rfloor$

```
ll mm = min(m, n);
for(ll i = 1, j; i <= mm; i = j + 1) {
    j = min(m / (m / i), n / (n / i));
    ans += (m / i) * (n / i) * (pre[j] - pre[i - 1]);
}
```

数论分块 证明

- 1. 块的数量：
  - 当 $i \le \sqrt{n}$ 时，显然至多$\lfloor\sqrt{n}\rfloor$种取值
  - 当 $i > \sqrt{n}$ 时，因$\left\lfloor\frac{n}{i}\right\rfloor \le \lfloor\sqrt{n}\rfloor$，也至多只有$\lfloor\sqrt{n}\rfloor$种取值
  - 这是一个比较重要的思想
- 2. 块中最大下标：
  - $\lfloor n/i\rfloor = \lfloor n/j\rfloor \Leftrightarrow \frac{n}{j} = d + \frac{\mathrm{r}}{\mathrm{j}}, 0 \le \mathrm{r} < \mathrm{j}, d = \lfloor n/i\rfloor$（带余除法）

$$\Leftrightarrow \frac{\mathrm{n}}{d} = \mathrm{j} + \frac{\mathrm{r}}{\mathrm{d}} \Leftrightarrow \frac{\mathrm{n-j}}{\mathrm{d}} < \mathrm{j} \le \frac{\mathrm{n}}{\mathrm{d}}$$

  - 又因为j为整数，故$j_{max} = \left\lfloor\frac{n}{d}\right\rfloor = \left\lfloor\frac{n}{\lfloor n/i\rfloor}\right\rfloor$

排列组合

- 组合数的计算 / 预处理：通常来说直接预处理阶乘以及阶乘逆元然后代公式即可。（其实阶乘逆元最好应该用线性求[1, n]逆元的方法解决，但我有时太懒就直接全用快速幂了。。。）
- 组合数公式（推式子常用）：
  1.组合数之间的递推公式
  2.二项式定理->正用与逆用，求导 / 求积分
- 组合计数：比较基础的计数问题：采取正确的计数策略（乘法/加法/容斥原理，从哪个方向入手计算贡献…），常规计数方法（隔板法等），组合公式推导（二项式定理等）

Fibonacci另一种求法

- $F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} mod\ p$，$p$为素数
- 公式中带有$\sqrt{5}$，为什么计算得到的F_n是一个整数?
- 二项式展开，发现$\sqrt{5}$的奇数次幂在展开式中没有出现
- 所以只需要找到一个整数$x : x^2 \equiv 5\ (mod\ p)$
- 用x替换$\sqrt{5}$，结果一定是相同的，而且在计算过程中也可以用逆元和快速幂了。
- x又称作 5 模 p 的**二次剩余**