一些数学知识和算法

2014.6

卷积及其应用

2014.6

圆周卷积

- 序列 x 和 y 是周期为 N 的序列：

$$x_k = x_{k \bmod N}$$

- x 和 y 的卷积即为

$$(x \otimes y)_n = \sum_{m=0}^{N-1} x_m y_{n-m}$$

乘法即为卷积

- 如果 x 和 y 表示的均为高精度数，则

$$(x \times y)_n = \sum_{m=0}^{n} x_m y_{n-m}$$

$x \times y$ 可以看作周期是 $2N$ 的序列。

- 这二者有微妙的不同。但是如果令 $\tilde{x}$ 表示在 x 后面补上 n 个 0 得到的周期为 $2n$ 的序列，即

$$\tilde{x}_k = \begin{cases} x_k & (k < n) \\ 0 & (n \le k < 2n) \end{cases}$$

那么 $x \times y = \tilde{x} \otimes \tilde{y}$。

FFT

- 长度为 2^k 的卷积的计算可以用快速傅里叶变换进行，复杂度为 $O(N\log N)$。
- 记 $\omega_N = e^{\frac{-2\pi i}{N}}$，即 n 次单位根。
- 则序列 x 的离散傅里叶变换结果为

$$(\mathcal{F}x)_k = \sum_{n=0}^{N-1} \omega_N^{nk} x_n$$

FFT (cont'd)

- 它的逆变换为

$$(\mathcal{F}^{-1}x)_k = \frac{1}{N}\sum_{n=0}^{N-1} \omega_N^{-nk} x_k$$

- $\frac{1}{N}$ 是归一系数。两个式子中的归一系数可以均为 $\frac{1}{\sqrt{N}}$。

FFT (cont'd)

- 记 E 为 x 的偶子序列，即 $E_t = x_{2t}$。
- O 为 x 的奇子序列，即 $O_t = x_{2t+1}$。
- 那么有

$$(\mathcal{F}x)_k = \sum_{n=0}^{N-1} \omega_N^{kn} x_n = \sum_{n=2t} \omega_N^{kn} x_n + \sum_{n=2t+1} \omega_N^{kn} x_n \ (t \in \mathbb{Z})$$

$$= \sum_t \omega_N^{2kt} x_{2t} + \omega_N^k \sum_t \omega_N^{2kt} x_{2t+1}$$

而 $\omega_N^{2k} = \omega_{\frac{N}{2}}^{k}$，所以原式

$$= \sum_t \omega_{\frac{N}{2}}^{kt} E_t + \omega_N^k \sum_t \omega_{\frac{N}{2}}^{kt} O_t = (\mathcal{F}E)_k + \omega_N^k (\mathcal{F}O)_k$$

FFT (cont'd)

- 所以，计算离散傅里叶变换 (DFT)，可以先将序列拆成奇偶两部分，分别进行 DFT 之后再线性合并。
- 时间复杂度 $O(n\log n)$。
- 计算逆变换同理。

卷积定理

- 记序列 x 和 y 的积为对应元素的积的序列。即

$$(xy)_k = x_k y_k$$

- 那么有如下的卷积定理：

$$\mathcal{F}(x \otimes y) = \mathcal{F}x\mathcal{F}y$$

即

$$x \otimes y = \mathcal{F}^{-1}(\mathcal{F}x\mathcal{F}y)$$

卷积定理 (cont'd)

$$\mathcal{F}(x \otimes y)_n = \sum_{k=0}^{N-1} \omega_N^{nk} (x \otimes y)_k = \sum_{k=0}^{N-1} \omega_N^{nk} \sum_{m=0}^{N-1} x_m y_{k-m}$$

$$(\mathcal{F}x\mathcal{F}y)_n = \left(\sum_{k_0=0}^{N-1} \omega_N^{nk_0} x_{k_0} \right) \left(\sum_{k_1=0}^{N-1} \omega_N^{nk_1} y_{k_1} \right)$$

$$= \sum_{k_0=0}^{N-1} \sum_{k_1=0}^{N-1} \omega_N^{n(k_0+k_1)} x_{k_0} y_{k_1} \blacksquare$$

卷积的运算

- 由卷积定理我们可以得出计算卷积的有效方法：
  - 计算两个序列的 DFT……$O(N\log N)$
  - 将 DFT 直接相乘……$O(N)$
  - 计算乘积的 IDFT（DFT 的逆）……$O(N\log N)$
- 所以计算卷积的复杂度是 $O(N\log N)$。

大整数乘法

- 由于大整数乘法可以用圆周卷积表示，所以大整数乘法可以在 $O(N\log N)$ 的时间复杂度内完成。
- 需要注意的问题：
  - 至少在前面补出 N 个零。
  - 将长度扩宽成 2 的幂即可。

多项式乘法

- 两个 N 次多项式的积和大整数乘法是一样的。所以也可以在 $O(N\log N)$ 的时间复杂度内完成。
- 因此，我们可以广泛地使用生成函数这一有力工具。

生成函数

- 对于数列 a_n ，定义幂级数

$$A(x) = \sum_{k=0}^{\infty} a_n x^n$$

为 a_n 的生成函数。

- 这个幂级数不一定在 $x = 0$ 以外的点收敛。它只是一个形式。

生成函数的一些性质

- 若 $c_n = a_n + b_n$，则 $C(x) = A(x) + B(x)$。
- 若 $c_n = k^n a_n$，则 $C(x) = A(kx)$。
- 若 $c_n = na_n$，则 $C(x) = xA'(x)$。
- 若 $c_n = \sum_{k=0}^{n} a_k b_{n-k}$，则 $C(x) = A(x)B(x)$。
  - 作为上例的特殊情况，若 $c_n = \sum_{k=0}^{n} a_k$，则 $C(x) = A(x)(1 + x + x^2 + x^3 + \cdots) = \frac{A(x)}{1-x}$。

一个重要的式子

$$1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x}$$

- 这是 $a_n = 1$ 的生成函数。
- 利用前面的性质可以得到 $a_n = k^n$ 的生成函数即为 $\frac{1}{1-kx}$。

生成函数作为解数列递推式的工具

- 如果需要解出数列的通项，可以考虑生成函数。
- 方法是，先解出生成函数 $f(x)$，再用 Taylor 展开或其他方法将其表示成幂级数的形式。

例子：线性递推

- 解递推式

$$\begin{cases} a_n - 5a_{n-1} + 6a_{n-2} = 0 \\ a_0 = 1, a_1 = -2 \end{cases}$$

例子：线性递推 (cont’d)

- 设 $A(x)=\sum a_n x^n$，则

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots$$

$$-5xA(x) = 0 - 5a_0x - 5a_1x^2 - 5a_2x^3 + \cdots$$

$$6x^2A(x) = 0 + 0x + 6a_0x^2 + 6a_1x^3 + \cdots$$

- 将三式相加，得到

$$(1-5x+6x^2)A(x) = a_0 + (a_1 - 5a_0)x = 1 - 7x$$

- 从而

$$A(x) = \frac{1-7x}{1-5x+6x^2}$$

例子：线性递推 (cont'd)

- 利用 Taylor 展开，或者部分分式分解

$$\frac{1-7x}{1-5x+6x^2}=\frac{5}{1-2x}-\frac{4}{1-3x}$$

可以得到

$$a_n=5\cdot 2^n-4\cdot 3^n$$

- （当然，也可以用特征方程的方式求解）

例子：Catalan 数

- 解递推式

$$\begin{cases} h_n = \sum_{k=1}^{n-1} h_k h_{n-k} \ (n \geq 2) \\ h_1 = 1 \end{cases}$$

例子：Catalan 数 (cont'd)

- 设 $H(x)=\sum h_n x^n$。
- 将 $H(x)$ 平方，得到

$$H^2(x)=\sum_{n=2}^{\infty} x^n \sum_{k=1}^{n-1} h_k h_{n-k} = H(x)-x$$

例子：Catalan 数 (cont'd)

- 解一元二次方程得到

$$H(x) = \frac{1 \pm \sqrt{1-4x}}{2}$$

- 而 $H(0) = 0$，应该取负号。所以

$$H(x) = \frac{1 - \sqrt{1-4x}}{2} = \frac{1}{2} - \frac{1}{2}(1-4x)^{\frac{1}{2}}$$

- 将其 Taylor 展开得到

$$h_n = -\frac{1}{2}(-1)^n 4^n \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\ldots\left(\frac{1}{2}-n+1\right)}{n!}$$

$$= -\frac{1}{2}(-4)^n \frac{(-1)^{n-1} 1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2n-3)}{2^n n!}$$

$$= -\frac{1}{2}(-4)^n \frac{(-1)^{n-1}(2n-2)!}{2^n n!\, 2^{n-1}(n-1)!} = \frac{1}{n}\binom{2n-2}{n-1}$$

多重集组合数

- 考虑多重集 $S = S_1 \cup S_2$。
  - 在 S 中取 n 个数分两步，先在 S_1 中取 k 个数，再在 S_2 中取 $n-k$ 个数。
  - 设在 S, S_1, S_2 中无序取 n 个数的方案数分别为 c_n, a_n, b_n ，则

$$c_n = \sum_{k=0}^{n} a_k b_{n-k}$$

  - 由生成函数的性质有 $C(x) = A(x)B(x)$。

多重集组合数 (cont’d)

- 考虑多重集 $\{\infty \cdot s_1\}$。
  - 生成函数 $A(x) = \frac{1}{1-x}$。
- 那么多重集 $\{\infty \cdot s_1, \infty \cdot s_2, \ldots, \infty \cdot s_n\}$ 呢？
  - $A(x) = \frac{1}{(1-x)^n} = \sum_{r=0}^{\infty} \binom{n+r-1}{r} x^r$（牛顿二项式定理）。
  - 与隔板法得到的结果相同。

多重集组合数 (cont’d)

- 考虑多重集 $\{\infty \cdot s_1\}$ ，但是 s_1 只能取 3 的倍数次。
  - 生成函数 $A(x) = 1 + x^3 + x^6 + x^9 + x^{12} + \cdots$
- 考虑多重集 $\{\infty \cdot s_1\}$，但是 s_1 只能取素数次。
  - 生成函数 $A(x) = x^2 + x^3 + x^5 + x^7 + x^{11} + \cdots$
- 考虑多重集 $\{5 \cdot s_1\}$。
  - 生成函数 $A(x) = 1 + x + x^2 + x^3 + x^4 + x^5$

多重集组合数 (cont'd)

- 考虑多重集 $\{\infty \cdot s_1, \infty \cdot s_2, 5 \cdot s_3\}$，但是 s_1 只能取 3 的倍数次，s_2 只能取素数次。
  - 生成函数 $A(x) = (1 + x^3 + x^6 + x^9 + \ldots)(x^2 + x^3 + x^5 + x^7 + x^{11} + \ldots)(1 + x + x^2 + x^3 + x^4 + x^5)$
- 也就是说，正常的多重集组合数问题都可以用生成函数解决。

多重集组合数 (cont'd)

- 如此得到的每个多项式的次数一般都比较高。此类问题需要 FFT 来解决。

正整数剖分问题

- 考虑如下问题：
  - 对 N 进行无序的允许重复的剖分，求方案数。
  - 对 N 进行无序的不允许重复的剖分，求方案数。

正整数剖分问题

- 对 N 进行无序的允许重复的剖分，求方案数。
  - 生成函数

    $$(1+x+x^2+x^3+\cdots)(1+x^2+x^4+x^6+\cdots)(1+x^3+x^6+x^9+\cdots)\ldots$$
  - 即

    $$\frac{1}{(1-x)(1-x^2)(1-x^3)\ldots(1-x^n)\ldots}$$

正整数剖分问题

- 对 N 进行无序的不允许重复的剖分，求方案数。
  - 生成函数 $(1+x)(1+x^2)(1+x^3)\ldots$
- 类似这样的思路，下面的问题都类似。
  - 把 N 无序剖分成素数的和，允许/不允许重复。
  - 把 N 无序剖分，每个数最多重复 2 次。
  - ……

指数生成函数

- 数列 a_n 的指数生成函数 $A_e(x)$ 为

$$A_e(x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$$

- 它有性质

$$C_e(x) = A_e(x)B_e(x) \Leftrightarrow c_n = \sum_{k=0}^{n} \binom{n}{k} a_k b_{n-k}$$

指数生成函数 (cont'd)

- 类似于前面生成函数的用法，用指数生成函数可以解决多重集的排列问题。
- 但是由于系数中有分母 $n!$，所以并不好用。
- 仅供参考。

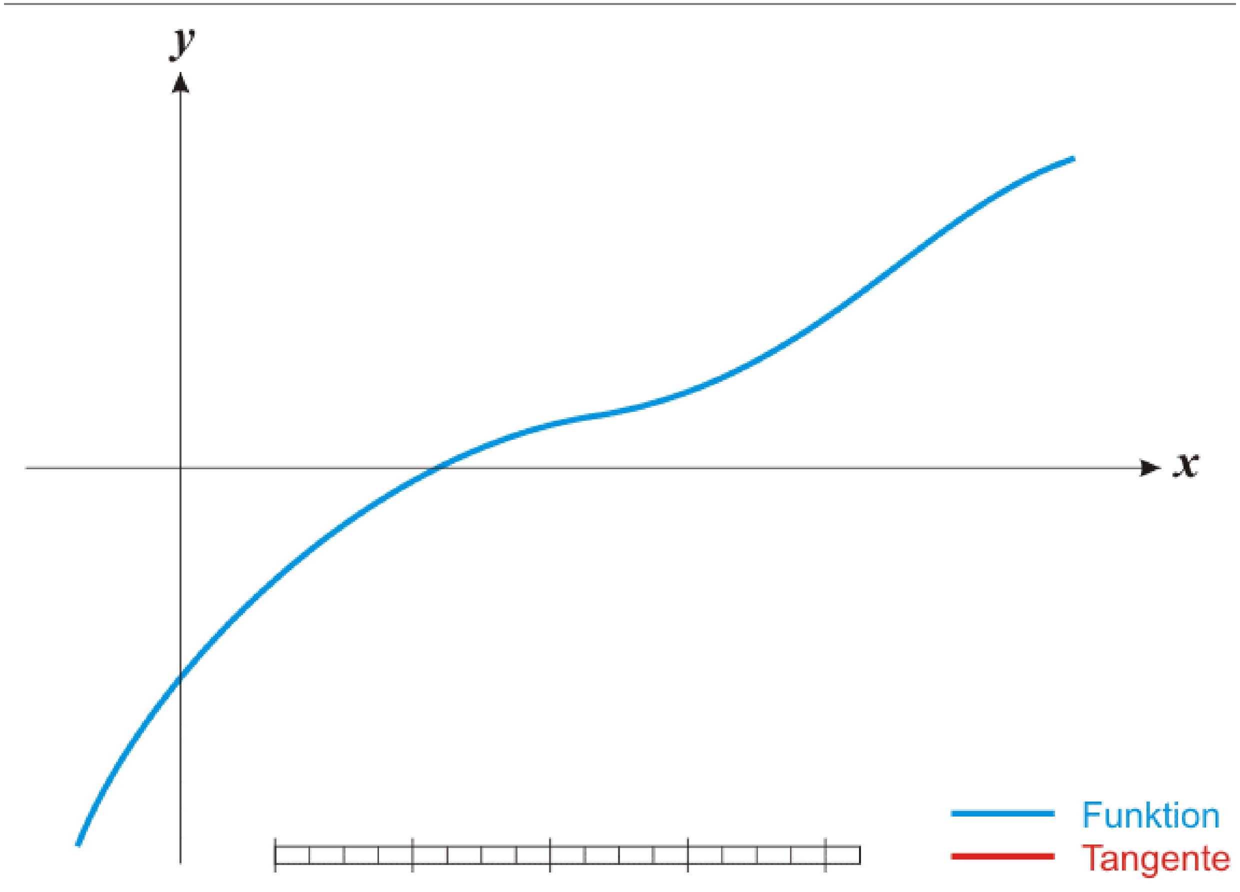
牛顿迭代法

- 求方程 $f(x) = 0$ 的根的方法。
- 首先，选择一个接近函数零点的 x_0 ，计算 $f(x_0)$ 和 $f'(x_0)$。
- 然后计算穿过点 $(x_0, f(x_0))$ 并且斜率为 $f'(x_0)$ 的直线与 x 轴的交点的横坐标 x_1。
- 用 x_1 取代 x_0 ，开始新的迭代。

首先，选择一个接近函数零点的 x_0，计算 $f(x_0)$ 和 $f'(x_0)$。

然后计算穿过点 $(x_0, f(x_0))$ 并且斜率为 $f'(x_0)$ 的直线与 x 轴的交点的横坐标 x_1。

用 x_1 取代 x_0，开始新的迭代。



牛顿迭代法（via wiki）

牛顿迭代法 (cont’d)

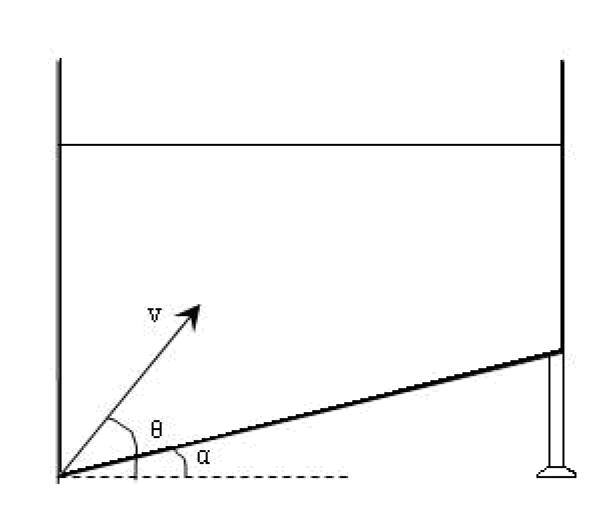
- 迭代公式可做如下简化：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- 如果 f' 是连续的，并且零点 x 是孤立的，那么存在 x 的一个邻域，使得初始值在这个邻域中时，迭代法收敛。
- 并且，如果 f' 非零，牛顿迭代法具有平方收敛的性能。

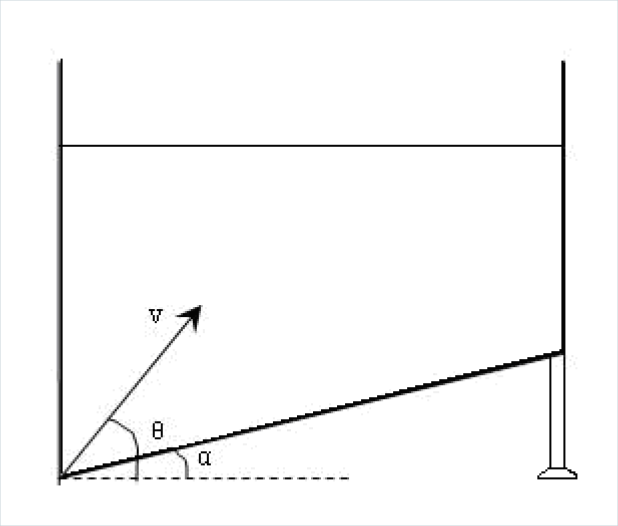
例子：New Liquid

- 我们有一缸神奇液体，这种液体的密度无限接近于零，却能对在其中运动的物体施加阻力，使得物体得到一个 $f\vec{v}$ 的加速度，其中 $f<0$。



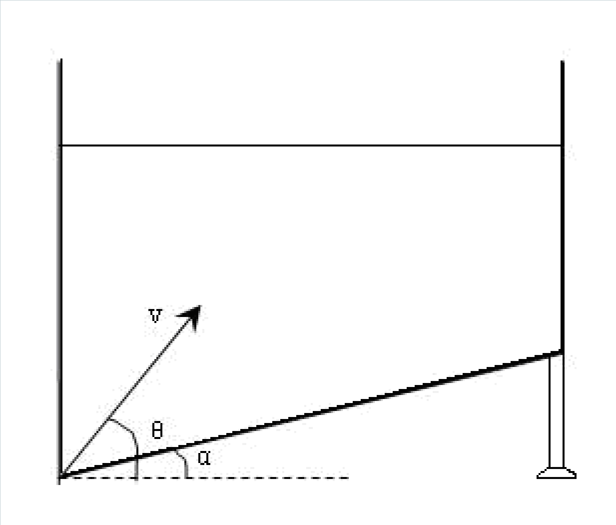
例子：New Liquid (cont'd)

- 而缸的底面与地面有着 α 的夹角。现在我们要从缸的最底下扔一个球，初速率为 v，问初速度的方向与地面的夹角 θ 为何值时，球砸在缸底的那一点与投掷点的距离最大。



例子：New Liquid (cont'd)

- 注意，θ 是一个整数角度，你只要求出所有的整数角度中的最优值。而且，由于密度接近 0，你不需要考虑浮力带来的影响。



例子：New Liquid (cont’d)

- 以投掷点为原点，地面为 x 轴，缸的左壁为 y 轴建立平面直角坐标系。
- 初速度 $\overrightarrow{v_0} = (v\cos\theta, v\sin\theta)$。
- 加速度 $\vec{a} = f\vec{v} + \vec{g}$，那么 $\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = f\vec{v} + \vec{g}$。
- 在两个维度上解线性微分方程，得到

$$\vec{v}(t) = \left(v\cos\theta\, e^{ft}, v\sin\theta\, e^{ft} + \frac{g}{f}\left(e^{ft} - 1\right)\right)$$

例子：New Liquid (cont’d)

- 位移

$$\vec{s} = \int \vec{v}\mathrm{d}t$$

$$= \left(\frac{v\cos\theta\left(e^{ft}-1\right)}{f}, \frac{(vf\sin\theta+g)\left(e^{ft}-1\right)}{f^2} + \frac{gt}{f} \right)$$

例子：New Liquid (cont’d)

- 如果 t_0 时间与缸底相碰，那么 $< \vec{s}(t_0), \vec{x} > = \alpha$。
- 代入化简，得到

$$\frac{t_0}{e^{f t_0} - 1} = \frac{v \cos\theta\,(\tan\alpha - \tan\theta)}{g} + \frac{1}{f}$$

- 右面是个常数。即解方程 $\frac{x}{e^{fx} - 1} - C = 0$。

牛顿迭代法的作用

- 有了牛顿迭代法，我们在解决问题的时候就少了一道阻碍，只要函数的导数连续且非零，我们就可以直接对其求解。

注意事项

- 初值的选取。
  - 距离待求零点越接近越好。否则可能会减慢速度，甚至出现不收敛的情况。
- 是否收敛？
  - 理论分析。
  - 实际测试。

大数除法

- 如何做大数除法？
  - 算法一：模拟
    - 时间复杂度 $O(n^2)$
  - 算法二：直接牛顿迭代
    - 时间复杂度 $O(n\log^2 n)$
  - 算法三：倍增法
    - 时间复杂度 $O(n\log n)$

牛顿迭代做除法

- a 的倒数即为 $f(x)=\frac{1}{x}-a$ 的零点。
- $x_{n+1}=x_n-\frac{\frac{1}{x_n}-a}{-\frac{1}{x_n^2}}=x_n+x_n-ax_n^2=2x_n-ax_n^2$
- 每次迭代时间复杂度为 $O(n\log n)$。

牛顿迭代做除法 (cont'd)

- 相对误差：

$$\epsilon_n \overset{\text{def}}{=} a\left(\frac{1}{a} - x_n\right) = 1 - ax_n$$

$$= 1 - a(2x_{n-1} - ax_{n-1}^2) = 1 - 2ax_{n-1} + a^2x_{n-1}^2$$

$$= (1 - ax_{n-1})^2 = \epsilon_{n-1}^2$$

- 所以初始值的相对误差在 1 以内即可收敛。
- 迭代次数为 $O(\log n)$，所以总的时间复杂度为 $O(n \log^2 n)$。

倍增法做除法

- 假设要计算的是 $\frac{b}{a}$，其中 b 是 $2n$ 位的整数，a 是 n 位的整数。
- 若有 n 位整数 $c = \frac{10^{2n}}{a}$，则 $\frac{b}{a} = \frac{bc}{10^{2n}}$（注意，有精度误差）。
- 计算 c 即可。所以现在的问题是如何计算 c。

倍增法做除法 (cont'd)

- 计算 $\frac{10^{2n}}{a}$，即为计算 $f(x) = \frac{1}{x} - \frac{a}{10^{2n}}$ 的零点。所以迭代式为 $x_{n+1} = -\frac{a}{x^{2n}} x_n^2 + 2x_n$.
- 相对误差为 $1 - \frac{a}{10^{2n}} x_n$.

倍增法做除法 (cont’d)

- 使用一个特别的技巧，令 c 为 a 的前 $\left\lceil \frac{n}{2} \right\rceil$ 位，即

  $a = c \cdot 10^{n-t} + p_0$，其中 $t = \left\lceil \frac{n}{2} \right\rceil, p_0 < 10^{n-t}$。

- 假设我们已经计算出了 d，使得 $cd = 10^{2t} + p_1$，其中 $p_1 < 10^t$，那么相对误差

$$1 - \frac{c}{10^{2t}} d = \frac{p_1}{10^{2t}} < 10^{-t}$$

倍增法做除法 (cont'd)

- 令 $x_0 = 10^{n-t}d$ 为迭代初值。它的相对误差为

$$1 - \frac{a}{10^{2n}}x_0 = 1 - \frac{10^{n-t}c + p_0}{10^{2n}}d \cdot 10^{n-t}$$

$$= 1 - 10^{-2t}cd + 10^{-2n}p_0 d$$

$$= 1 - 10^{-2t}(10^{2t} + p_1) + 10^{-2n}p_0 d$$

$$= 10^{-2t}p_1 + 10^{-2n}p_0 d$$

$$< 10^{-t} + 10^{-n}$$

倍增法做除法 (cont'd)

- 一次迭代之后，相对误差的级别为 $10^{-2t} < 10^{-n}$。
- 所以先减半，算出 d 之后一次迭代即可得到结果。
- 注意绝对误差可能仍然有 10，所以需要 $n+1$ 位除以 n 位的朴素除法进行调整。

倍增法做除法 (cont’d)

- 复杂度方面，算法依次处理了 $n, \frac{n}{2}, \frac{n}{4}, \ldots$ 长度的整数，所以复杂度为 $O(n\log n) + O\left(\frac{n}{2}\log n\right) + O\left(\frac{n}{4}\log n\right) + \cdots = O(2n\log n) = O(n\log n)$。
- 实现上常数也不大。所以速度很快。

多项式除法

- 多项式的除法也可以用牛顿迭代法完成。
- 类似于前面的倍增法，将以 10 为基换成以 x 为基，既可以得到做法。
- 而且无需调整。

多项式的牛顿迭代

- 将高精度数的牛顿迭代中的基数，形式化地写成 x，就可以对多项式进行牛顿迭代了。
- 用类似的方法，可以求一些神乎其神的东西。

例：Bank Craft

- 给定一个 n 次以内的多项式 $P(x)$，求 n 次以内的多项式 $Q(x)$ 使得存在 $R(x)$ 满足

$$P(x)Q(x) = 1 + x^n R(x)$$

例：Bank Craft (cont'd)

- 原式等价于

$$P\left(\frac{1}{x}\right)Q\left(\frac{1}{x}\right)=1+x^{-n}R\left(\frac{1}{x}\right)$$

两边乘以 x^{2n}，得到

$$\tilde{P}(x)\tilde{Q}(x)=x^{2n}+\tilde{R}(x)$$

其中 $\tilde{P}(x)$ 表示将 $P(x)$ 的系数按 n 次多项式颠倒。

例：Bank Craft (cont'd)

- 此即

$$\tilde{Q}(x) = \frac{x^{2n} + \tilde{R}(x)}{\tilde{P}(x)}$$

- $\tilde{P}(x)$ 的最高次项 (即原先的常数项) 若为 0 则无解。
- 否则求 $\frac{x^{2n}}{\tilde{P}(x)}$ 的商式即可。

数论转换

- FFT 需要在双精度浮点数上做，会导致精度问题。
- 究其原因，是因为单位根是复数。
- 若能够寻求单位根的替代品，问题就解决了。

数论转换 (cont’d)

- 这个替代品需要具有以下的性质：
  - $\omega_N^N = 1$，且 $\omega_N^p \neq 1(p \neq kN, k \in \mathbb{Z})$。
  - $\omega_N^{k+\frac{N}{2}} = -\omega_N^k(N = 2k, k \in \mathbb{Z})$。
  - $\omega_N^{pq} = \omega_{N/p}^q(N = pk, k \in \mathbb{Z})$。

数论转换 (cont’d)

- 若整数 a 满足 $\delta_M(a) = N$，则称 a 是 M 的 N 次单位根。
- 如果 M 是素数，容易验证在模 M 的情况下 a 满足上面的全部三条性质。
- 所以我们可以用 a 来代替 ω。

数论转换 (cont'd)

- 数论转换的转换式水到渠成：

$$\hat{x}_k = \sum_{n=0}^{N-1} a^{nk} x_n \pmod{M}$$

- 其逆转换也和原来的式子十分类似：

$$x_n = N^{-1} \sum_{k=0}^{N-1} a^{-nk} \hat{x}_k \pmod{M}$$

数论转换 (cont’d)

- 各个变量要满足以下条件：
  - M 是一个素数。
  - $\delta_M(a) = N \Rightarrow N|(M-1)$。
  - $N = 2^r \Rightarrow M = k \cdot 2^r + 1$。FFT 要求长度是 2 的幂。
  - 卷积的结果是模 M 的。 M 太小结果会错，太大会溢出。
- 单位根取 $g^{\frac{M-1}{N}}$ 即可。其中 g 是 M 的原根。

数论转换 (cont'd)

- 整数圆周卷积专用算法。
- 没有精度误差。
- 速度较慢。（double 类型的乘法远远快于整数取模）

其他内容

从我原来的课件上粘过来的内容

真·扩展欧几里得算法

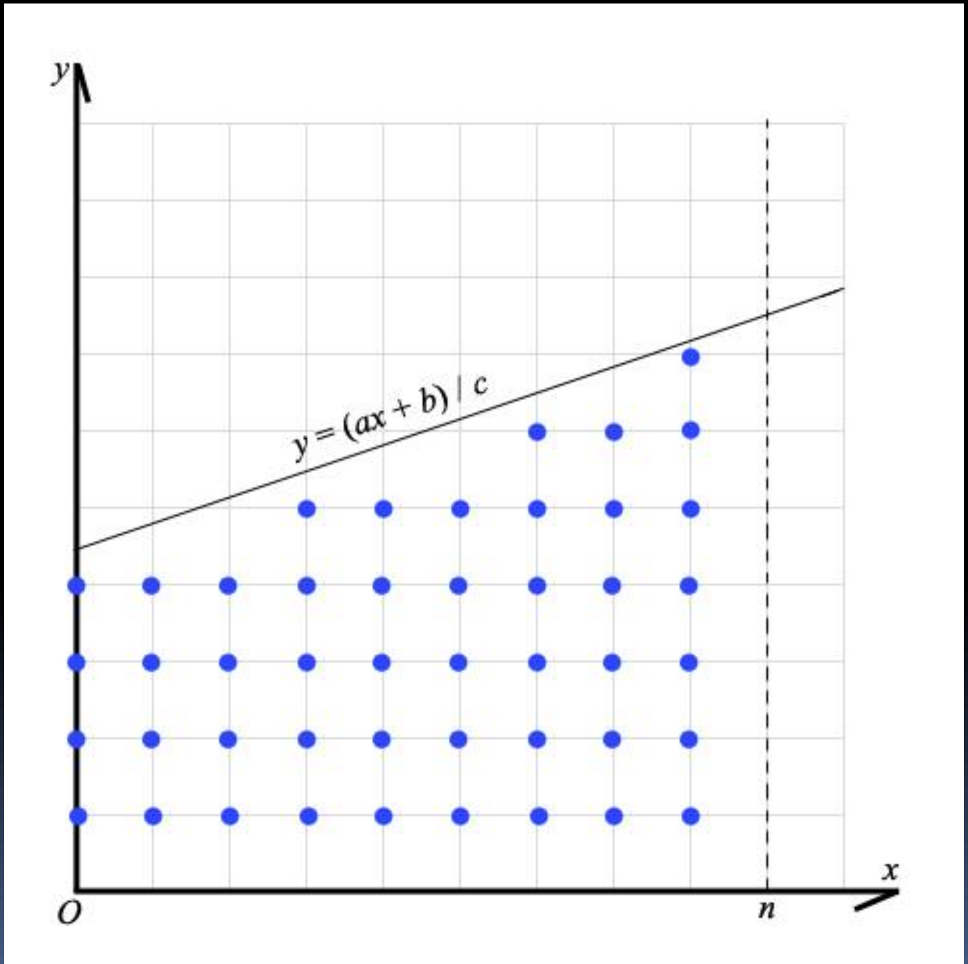
- 计算

$$\sum_{x=0}^{n-1}\left\lfloor\frac{ax+b}{c}\right\rfloor$$

的值。

- 我用剩下的不多的那点节操向你保证 a, b, c 是整数，且 $a, b, c, n \le 10^9$。

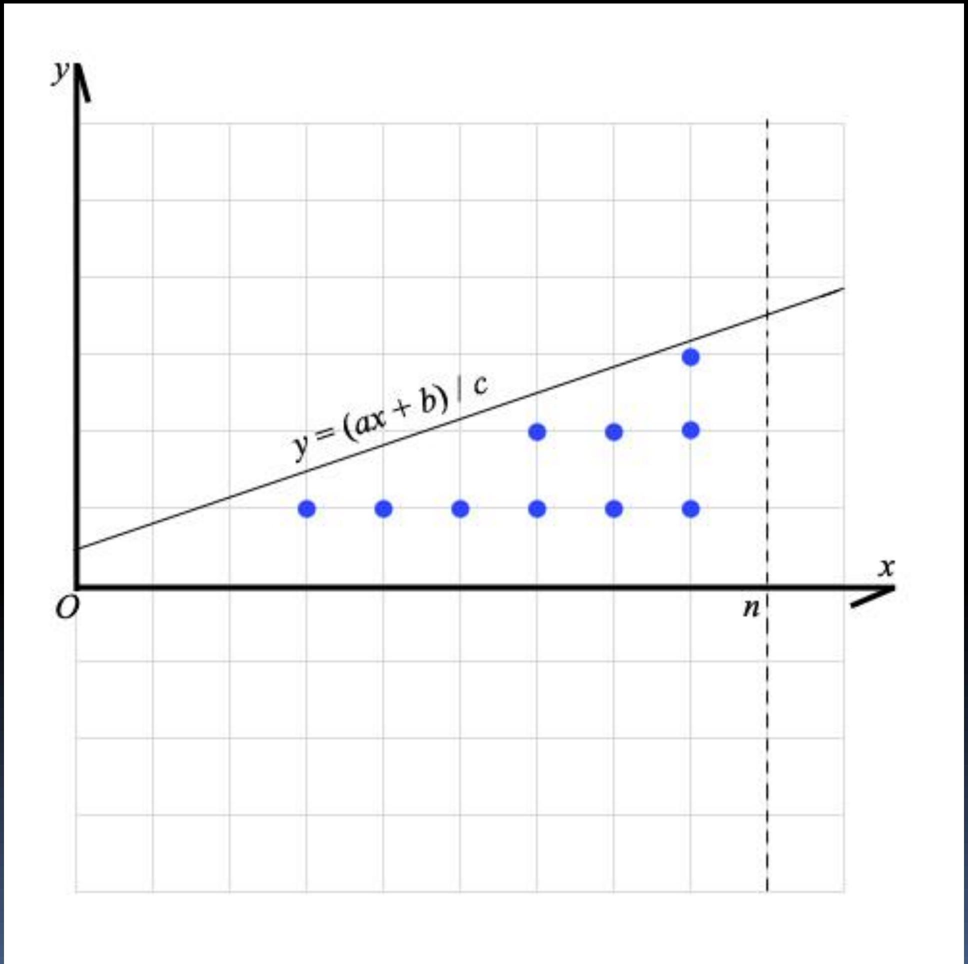
你要算的是这个



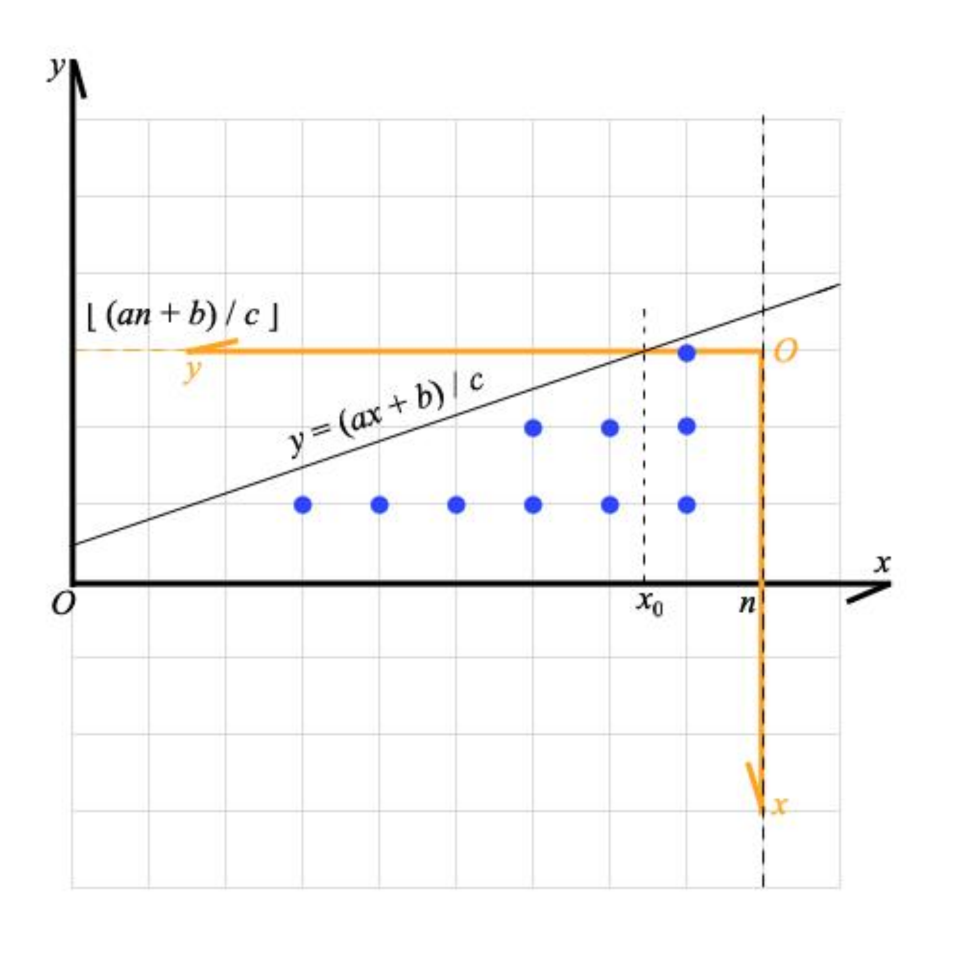
扫清障碍

- 若 $a = 0$，那这事就搞定了。
- 否则，通过在取整号里加上一些整数把 a 化到 $(0, c)$ 的范围中。
- 于是这条直线的斜率小于 1。
- 同样地，把 b 化到 $[0, c)$ 的范围中。

然后就成了这个



你的目的是这个



坐标转化

- 建立一个原点在$\left(n,\left\lfloor\frac{an+b}{c}\right\rfloor\right)$，以原 $-y$ 轴为 x 轴，$-x$ 轴为 y 轴的新坐标系。
- 在这个坐标系中，直线的斜率大于 1。
- 于是就可以递归算了。
- 但是这条直线在新坐标系中的方程是什么？

求解直线方程

- 在新坐标系中显然直线的斜率为原来的倒数，即 $\frac{c}{a}$。
- 而 x_0 满足 $\frac{ax_0+b}{c}=\left\lfloor\frac{an+b}{c}\right\rfloor$。
- 所以其 y 截距为

$$n-x_0=n-\frac{\left\lfloor\frac{an+b}{c}\right\rfloor c-b}{a}$$

$$=\frac{an+b-\left\lfloor\frac{an+b}{c}\right\rfloor c}{a}=\frac{(an+b)\ mod\ c}{a}。$$

递归

- 所以当 $a \in (0, c), b \in [0, c)$ 时，

$$\sum_{x=0}^{n-1} \left\lfloor \frac{ax+b}{c} \right\rfloor = \sum_{x=0}^{\left\lfloor \frac{an+b}{c} \right\rfloor - 1} \left\lfloor \frac{cx + (an+b) \bmod c}{a} \right\rfloor$$

- 递归运算即可。
- 时间复杂度和欧几里得算法一样是 O(log c)。

例子： Bitwise XOR of Arithmetic Progression

- 对于正整数 x, y, z ($x, y, z \le 2^{32}, x \le y$)，计算 x XOR $(x + z)$ XOR $(x + 2z)$ XOR … XOR $(x + uz)$ 的值。
- 其中 u 为满足 $x + uz \le y$ 的最大的 u。

解答

- 对于某个 $x+kz$，若 $(x+kz)$ AND $(2^P)=1$，则 $\left\lfloor\frac{x+kz}{2^P}\right\rfloor$ 为奇数。
- 若答案此位为 1，则 $\sum_{k=0}^{u}\left\lfloor\frac{x+kz}{2^P}\right\rfloor$ 为奇数。
- 计算这个式子的复杂度为 $\log y$。而一共有 $\log y$ 位需要计算，所以复杂度为 $O(\log^2 y)$。