

# Discrete

## Mathematics - P35

8.9 - 矩阵乘法

~~矩阵乘法~~ 两个矩阵乘积的定义可以表达为计算两个矩阵乘积的算法

另外同样的算法可用于描述两个矩阵的布尔积。

procedure matrix multiplication (矩阵 A, B)      procedure Boolean product (0-1 矩阵 A, B)

for i := 1 to m

for j := 1 to n

$C_{ij} := 0$

for q := 1 to k

$C_{ij} := C_{ij} + a_{iq} b_{qj}$

return C      { $k \times n$  矩阵 B 的乘积  
 $\{m \times n$  矩阵 C 是  $m \times k$  矩阵 A 与 }

for i := 1 to m

for j := 1 to n

$C_{ij} := 0$

for q := 1 to k

$C_{ij} = C_{ij} \vee (a_{iq} \wedge b_{qj})$

return C      { $m \times n$  0-1 矩阵 C 是  
 $m \times k$  0-1 矩阵 A 与  $k \times n$  0-1 矩阵 B 的布尔积}

注意:  $m \times k$  矩阵与  $k \times n$  矩阵相乘 / 布尔积时, 需要做  $m \times k \times n$  次整数乘法 / 按位与运算

## 矩阵链乘法

对于矩阵链  $A_1, A_2, \dots, A_n$ , 其中  $A_1, A_2, \dots, A_n$  分别为  $m_1 \times m_2, m_2 \times m_3, \dots, m_n \times m_1$  的矩阵

由于矩阵乘法是可结合的, 则如何安排乘法结合顺序, 使得以最少的整数乘法次数计算矩阵链其过程可大体地描述为, 先对  $m_2, m_3, \dots, m_n$  进行重排序, 使得  $m'_2 \geq m'_3 \geq \dots \geq m'_n$

然后依序对原位置的矩阵进行乘法运算

## 算法范型

(algorithmic paradigm), 指基于一种特定概念的通用方法, 可以用来构造求解一类广泛问题的算法

## 暴力算法

(brute-force algorithm). 通过基于对问题的描述和术语的定义以最直接的方式解决的

注意通常设计暴力算法来解决不太在意所需计算资源的问题 (如时间复杂度)

如求解一个问题时通过检查每一种可能的解, 来找出最可能的解

## 最近点对

(closest pair of points), 即寻找平面上  $n$  个点的集合中距离最近的点。

procedure closest-pair (实数对  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \in \mathbb{R} \times \mathbb{R}$ )

$min := \infty$

for i := 2 to n

for j := 1 to i - 1

if  $(x_i - x_j)^2 + (y_i - y_j)^2 < min$  then

$min := (x_i - x_j)^2 + (y_i - y_j)^2$

closest-pair :=  $((x_i, y_i), (x_j, y_j))$

return closest-pair

最近点对的暴力算法使用  $\Theta(n^2)$  次操作, 而更有效率的算法复杂度为  $O(n \log n)$

# Discrete

## Mathematics - P36

复杂度

- 常量 (constant complexity), 表示为  $\Theta(1)$ , 如尾递归优化后的函数调用占用内存
- 对数 (logarithmic complexity), 表示为  $\Theta(\log n)$ , 如二分搜索算法
- 线性 (linear complexity), 表示为  $\Theta(n)$ , 如线性搜索算法
- 线性对数 (linearithmic complexity), 表示为  $\Theta(n \log n)$ , 如归并排序算法
- 多项式 (polynomial complexity), 表示为  $\Theta(n^b)$ , 如冒泡排序和插入排序算法为  $\Theta(n^2)$  的  
注意:  $b$  通常是大于 1 的整数, 即  $b > 1$  且  $b \in \mathbb{Z}$
- 指数 (exponential complexity), 表示为  $\Theta(b^n)$ , 其中  $b > 1$   
如判定  $n$  个变量的复合命题是否可满足的算法是  $\Theta(2^n)$  的
- 阶乘 (~~factorial complexity~~ factorial complexity), 表示为  $\Theta(n!)$ ,  
如列举一个推销员访问  $n$  个城市的所有顺序的算法是  $\Theta(n!)$  的

易解性 (tractability), 指能用多项式最坏情形复杂度的算法求解的问题, 称为易解的 (tractable)

针对问题在合理规模的输入下, 可期望算法在相对短的时间内给出解答

注意: 由于多项式的系数非常大或者次数过高, 则也不能保证在合理时间内得到解答

难解的 (intractable), 指不能用最坏情形多项式时间复杂度解决的问题

通常即使对于小规模的输入在最坏情形下也需要特别大量的时间来解决问题  
但实践中, 具有某种最坏情形复杂度的算法大多数情况下能够比最坏情形下更快地解决问题  
允许少量情况下问题不能在合理时间内得到解答时, 平均情形复杂度是更好的度量方式  
另一种方案是, 寻求问题是的近似解而非精确解  
也许存在求近似解的快速算法 (如贪婪算法), 甚至保证近似解和精确解相差不大

NP ~~类~~ (nondeterministic polynomial) 指非确定性多项式时间  
即对于可解的问题, 设有多项式最坏情形时间复杂度的算法能求解  
但是当存在一个解, 却可以用多项式时间内来验证  
能在多项式时间内验证正解的问题称为属于 NP 类问题, 而易解的问题属于 P 类

NP 完全问题 (NP-complete problem) 指一类具有特定性质的问题, 如可满足性问题

只要其中任何一个问题能用一个多项式时间最坏情形算法来求解

则 NP 类的所有问题是都能用多项式时间最坏情形算法来求解

Cook-Levin 定理指 NP 类的每个问题都可以通过确定性图灵机在多项式时间内归约为可满足性问题

# Discrete

009 - initial

## Mathematics - P37

P与NP问题 (P versus NP problem) 是指NP类问题是是否等于P类问题

如果  $P \neq NP$ , 则存在无法在多项式时间内求解但其解可以在多项式内验证的问题

目前仍未找到一个最坏情形多项式时间复杂度的算法解决任何NP完全问题

位串1的计数

procedure bit-count(位串 S)

count := 0 [初始化 count 为 0]

while S ≠ 0: [只要 S 不全为 0, 则每个循环将最右的一个变为 0, 并且 ++count]

count := count + 1 [如果 S 为形如 ...1000, 则 S-1 为 ...0111]

S := S  $\wedge$  (S-1) [注意 S 与 S-1 前半部分完全一致, 于是有 S  $\wedge$  (S-1) = ...0000]

return count {S 中 1 的个数} [即这一步将位串最右的一位 1 置为 0]

霍纳法

(Horner's method) 是一个有效计算多项式值的算法, 以乘法和加法次数来度量

计算  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  在  $x=c$  处的值

procedure Horner(c, a<sub>0</sub>, a<sub>1</sub>, ..., a<sub>n</sub> ∈ R)

y := a<sub>n</sub>

for i := 1 to n [循环 n 次, 总计 n 次加法, n 次乘法]

y := y \* c + a<sub>n-i</sub> [即多项式分解为 ((...((a<sub>n</sub>) \* c + a<sub>n-1</sub>) \* c + a<sub>n-2</sub>) ...) \* c + a<sub>1</sub>) \* c + a<sub>0</sub>]

return y

每一个算法解题时相对于某一运算来说是最优的, 如果不存在其他算法在解此题时使用更少的此种运算

三角矩阵

(upper triangular), 对于一个  $n \times n$  矩阵, 当  $i > j$  时,  $a_{ij} = 0$ ,

procedure triangular multiplication (三角矩阵 A, B)

for i := 1 to n

for j := i to n

c<sub>ij</sub> := 0

for k := i to j

c<sub>ij</sub> := c<sub>ij</sub> + a<sub>ik</sub> \* b<sub>kj</sub>

return C

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

(以元素相乘度量的复杂度为  $\sum_{i=1}^n \sum_{j=i}^n (j-i+1) = \frac{1}{6}n(n+1)(n+2) = \sum_{i=1}^n i(i+1)/2$ )

字符串匹配

在一个 n 个字符的串, 或称文本中寻找一个给定的 m 个字符的串 (或称为目标) 首次出现的位置

procedure match string (字符串 s, t)

for i := 0 to n-m

is = TRUE

for j := 0 to m-1

if s[i+j] ≠ t[j] then is := FALSE, break

if is then return i {首次出现的位置}

return "未找到"

若从某一位开始, 未查连该字符是否匹配

若不匹配则右移一位

# Discrete Mathematics - P38

489 - Median

剃刀排序 (Shaker sort) 或称双向冒泡排序

交替地从头到尾和从尾到头扫描列表，如相邻元素对为逆序则交换，直到不需要交换

procedure shaker sort ( $\{a_1, a_2, \dots, a_n\} \subset R$ )

for  $i := 0$  to  $\lfloor n/2 \rfloor - 1$  do [在每个循环之后第  $i$  个元素和第  $n-i$  个元素完成排序]

  count := 0; for  $j := i$  to  $n-2-i$  do [计数器 count 用于记录每轮扫描中交换的次数]

    if  $a_j > a_{j+1}$  then 交换  $a_j, a_{j+1}$ , count := count + 1 [分别从头至尾和从尾至头扫描列表]

    if  $a_{n-i} < a_{n-i-1}$  then 交换  $a_{n-i}, a_{n-i-1}$ , count := count + 1 [如果相邻元素顺序不对，则交换元素，且计数 +1]

  if count = 0 then break [如果循环中计数仍为 0 则退出循环]

return  $\{a_1, a_2, \dots, a_n\}$  [返回排序后的数组]

合法伴侣 (Valid partner) 一位女士与一位男士为合法伴侣，当存在一个稳定匹配使他们结为夫妻

男性最优的 (male optimal)，指匹配中每位男士都被指派了他喜好列表中最高的合法伴侣

女性最差的 (female pessimal)，指匹配中每位女士都被指派了她喜好列表中最低的合法伴侣

注意：在男士为求婚者，女士为被求婚者的延迟接受算法中，会得到男性最优且女性最差的稳定匹配  
而如果改为女士为求婚者，男士为被求婚者的延迟接受算法，则会得到男性最差的 (male pessimal) 且女性最优的 (female optimal) 的稳定匹配

稳定匹配可以推广到男女数量不相等的情况

则可修改延迟接受算法为，以数量较多的一方为求婚者，较少一方为被求婚者

每位求婚者如果上一轮被拒，则继续求婚，直到已被喜好列表上所有被求婚者拒绝

稳定匹配还可以推广到存在男女配对被禁止的情况

则可修改延迟接受算法为，在求婚开始前，将禁止的配对从双方喜好列表删除

对于在单处理器上调度  $n$  个作业的问题，如果从  $s_j$  时刻开始，则在  $e_j = s_j + t_j$  时刻完成

对于作业  $j$ ，处理器必须不间断地运行  $t_j$  时间，而其截止时刻为  $d_j$

拖延 (lateness) 用于衡量作业在截止时间后多久完成，即作业  $j$  的拖延为  $\max(0, e_j - d_j)$

注意：可以用贪婪算法，每次选取截止时刻最早的作业运行

且以截止时刻递增顺序的调度，总是产生一个作业最大拖延最小化的调度

宽松度 (slackness) 指作业截止时刻和所需运行时间的差，则运行时间  $t$ ，截止时刻  $d$  的宽松度为  $d - t$

注意：以作业所需时间递增序的调度和以宽松度递减序的调度均不保证产生一个

如： $t_1 = 5, d_1 = 20, t_2 = 25, d_2 = 30$  最大拖延最小化的调度

则上述调度为  $j_2 \rightarrow j_1$ ， $j_1$  有 10 的拖延，而最优为  $j_1 \rightarrow j_2$  无拖延

# Discrete

## Mathematics - P39

背包问题 (knapsack problem), 对于一个总容量为  $W$  的背包和  $n$  件物品, 第  $j$  件物品的质量是  $w_j$ , 求解问题  
寻求  $n$  件物品的一个子集, 使其具有不超过  $W$  的尽可能大的总质量

procedure knapsack brute ( $W \in \mathbb{R}^+$ ,  $\{w_1, w_2, \dots, w_n\} \subset \mathbb{R}^+$ ) [背包问题的暴力算法]  
 $\max := 0$  for  $i := 0$  to  $2^n - 1$  [遍历所有物品组合的可能]  
 $str := 0$   $m := 0$

for  $j := 1$  to  $n$  [如果位串的第  $j$  位为 1, 则将第  $j$  件物品放入背包]  
if  $i$  的二进制第  $j$  位为 1 then  $m := m + w_j$

if  $m > \max \wedge m \leq W$  then  $\max := m$ ,  $str := i$  [如果总质量更接近总容量, 则更新记录]

return  $str, \max$  {str 的第  $j$  位为 1 表示放入第  $j$  件物品}

跨度 (makespan) 对于一组  $p$  个处理器和  $n$  个作业, 第  $j$  个作业需要不间断地运行  $t_j$  时间,

而一个处理器一次只能运行一个作业, 处理器  $k$  的负载  $L_k$  是指派给其的所有作业运行时间的总和

跨度为所有处理器的最大负载, 负载均衡问题寻求一种指派使得跨度最小化

procedure makespan greedy ( $p$  正整数,  $\{t_1, t_2, \dots, t_n\} \in \mathbb{R}^+$ )

对  $n$  个作业重新排序使得  $t_1 \geq t_2 \geq \dots \geq t_n$ , 生成  $p$  项空 list  $L$

for  $j := 1$  to  $n$  [对于用时最长的作业, 从  $p$  个处理器中]

找到  $L_k$  使得  $L_k$  是  $L$  中最小值 [选择已指派负载最小的一个  $L_k$ , 将作业  $j$ ]

$L_k := L_k + t_j$  [指派给处理器  $k$ ]

return  $\{L_1, L_2, \dots, L_p\}$

渐近算法 (approximation algorithm), 最优化问题的渐进算法生成的解保证接近于最优解

假设最优化问题针对输入  $S$  寻求最大化  $F(x)$ , 其中  $F(x)$  是针对输入  $x$  的函数

如果一个算法总能找到输入  $T$ , 使得  $F(T) \leq c F(S)$ , 其中  $c$  是一个固定的正实数

则称该算法为该问题的一个  $c$  渐近算法

如 负载均衡问题的贪婪算法是一个  $2$  渐近算法.

令  $L^*$  为最小跨度, 则有  $L^* \geq \max_{j=1,2,\dots,n} t_j$  且  $L^* \geq \frac{1}{p} \sum_{j=1}^n t_j$

则对于输入  $T$  有  $F(T)$ , 其中一步将  $t_j$  指派给  $L_k$ , 从而形成  $F(T)$

可知在这一步之前  $L_k$  为最小负载且有  $L_k < \frac{1}{p} \sum_{j=1}^n t_j$

又  $t_j \leq \max_{j=1,2,\dots,n} t_j$ , 则  $F(T) = L_k + t_j < \frac{1}{p} \sum_{j=1}^n t_j + \max_{j=1,2,\dots,n} t_j$

$\leq L^* + L^* = 2F(S)$

即有跨度的负载均衡问题的贪婪算法是一个  $2$  渐近算法

注意: 渐近算法多可以描述为贪婪算法, 虽然不能保证得到最优解

但可以得到一个足够的接近的解

# Discrete

## Mathematics - P40

$a$  整除  $b$  (a divides b) 记为  $a|b$ . 对于整数  $a, b$  且  $a \neq 0$ , 如果存在整数  $c$  使得  $b = ac$ , 则  $a$  整除  $b$ .  
 即有对于  $a, b \in \mathbb{Z} \wedge a \neq 0$ ,  $a|b \Leftrightarrow \exists c (c \in \mathbb{Z} \wedge b = ac) \Leftrightarrow \frac{b}{a} \in \mathbb{Z}$   
 当有  $a|b$  时, 我们  $a$  是  $b$  的一个因子或除数(divisor),  $b$  是  $a$  的一个倍数(multiple)  
 而当  $a$  不能整除  $b$  时, 记为  $\boxed{a} \nmid b$ .

性质

对于整数  $a, b, c$ , 其中  $a \neq 0$ , 则

$(a|b \wedge a|c) \rightarrow a|(b+c)$ , 即整除是封闭的.

$a|b \rightarrow \forall c \in \mathbb{Z} a|bc$ , 即如果  $a$  整除  $b$ , 则  $a$  整除  $b$  的任意倍数.

$(a|b \wedge b|c) \rightarrow a|c$ , 即整除具有传递性, 注意这里要求  $b \neq 0$ .

并由此推出: 如果对于整数  $a, b, c$  且  $a \neq 0$ , 有  $a|b$  且  $a|c$

则  $\forall m, n (m \in \mathbb{Z} \wedge n \in \mathbb{Z} \rightarrow a|m(b+n))$

除法算法

(division algorithm). 对于整数  $a$  和正整数  $d$ , 存在唯一的整数  $q$  和  $0 \leq r < d$ , 使得  $a = dq + r$

即有  $\forall a \in \mathbb{Z}, d \in \mathbb{Z}^+ \exists! q, r \in \mathbb{Z} (0 \leq r < d \wedge a = dq + r)$

证明过程有, 对  $a \in \mathbb{Z} \wedge d \in \mathbb{Z}^+$ , 令集合  $S = \{x \in \mathbb{N} \mid x = a - dq, q \in \mathbb{Z}\}$ .

可知集合非空且集合  $S$  中存在最小元素  $r$  (根据集合良序性), 且  $r = a - dq_0$ ,  $0 \leq r < d$ .

如果  $r_0 = \min S$ , 则令  $r_0 = a - dq_0 + 1 = a - dq_0 - d = r - d \geq 0$

即  $r_0 \in S$  且  $r_0 < r$ , 与  $r$  是最小元素矛盾.

所以可知存在整数  $r, q$  使得  $a = dq + r$  且  $0 \leq r < d$ .

另外如果存在整数  $r', q'$  使得  $a = dq' + r'$  且  $0 \leq r' < d$  且  $r \neq r', q \neq q'$

则  $dq + r = dq' + r'$ , 有  $r - r' = -d(q - q')$ .

又  $0 \leq r < d \wedge 0 \leq r' < d$ , 则  $-d < r - r' < d$ , 即  $-1 < q - q' < 1$

又  $q, q' \in \mathbb{Z}$ , 则  $q - q' = 0$ , 与  $q \neq q'$  矛盾.

所以可知存在唯一的整数  $r$  和  $q$ , 使得  $0 \leq r < d$  且  $a = dq + r$ .

注意这个证明过程利用了正整数集合的良序原理.

商 (quotient). 记为  $q = a \text{ div } d$ , 其中数字  $a$  为被除数(dividend),  $d$  为除数(divisor).

余数 (remainder) 记为  $r = a \text{ mod } d$ , 当  $a \in \mathbb{Z}$  且  $d \in \mathbb{Z}^+$  时, 有  $a \text{ div } d = \lfloor a/d \rfloor$ ,  $a \text{ mod } d = a - d \cdot \lfloor a/d \rfloor$ .

特别注意: 在  $a < 0$  或  $d < 0$  时, 不同的编程语言有不同的实现.

在 C++ 中, 有  $|a \text{ div } d| = \lfloor |a|/d \rfloor$ , 然后根据  $a, d$  的正负号确定符号.  $a \text{ mod } d = a - d \cdot \lfloor a/d \rfloor$

在 Python 中, 有  $a \text{ div } d = \lfloor a/d \rfloor$ ,  $a \text{ mod } d = a - d \cdot \lfloor a/d \rfloor$ , 与数学上在  $d > 0$  时一致 ( $d < 0$  时)

在 Haskell 中, 有  $a \text{ div } d = \lfloor a/d \rfloor$ ,  $a \text{ mod } d = a - d \cdot \lfloor a/d \rfloor$ . 与 Python 一致.

在 Racket 中, 有  $|a \text{ div } d| = \lfloor |a|/d \rfloor$ . 符号随  $a, d$  确定.  $a \text{ mod } d = a - d \cdot \lfloor a/d \rfloor$ . 与 C++ 一致.

即有  $-10 \text{ div } 3 = -3$ ,  $-10 \text{ mod } 3 = -1$ ,  $10 \text{ div } -3 = -3$ ,  $10 \text{ mod } -3 = 1$ ,  $-10 \text{ div } -3 = 3$ ,  $-10 \text{ mod } -3 = -1$ .

# Discrete

## Mathematics - P41

Section 9

同余式 (congruence), 用记号  $a \equiv b \pmod{m}$  表示  $a$  模  $m$  同余  $b$ ,  
或者  $a$  和  $b$  模  $m$  同余, 其中  $m$  是同余式的模 (modulus)

$a$  和  $b$  模  $m$  同余 ( $a$  and  $b$  are congruent modulo  $m$ ), 称  $a$  和  $b$  模  $m$  同余  
如果对于整数  $a, b$  和正整数  $m$ , 有  $m$  整除  $a-b$  ( $m$  divides  $a-b$ ), 即  $m | (a-b)$   
即有对于  $a, b \in \mathbb{Z}, m \in \mathbb{Z}^+$ , 有  $a \equiv b \pmod{m} \leftrightarrow m | (a-b)$   
而如果  $a$  和  $b$  不是模  $m$  同余, 则记为  $a \not\equiv b \pmod{m}$

等价  $a \equiv b \pmod{m}$  当且仅当  $a \pmod{m} = b \pmod{m}$ , 即  $a \equiv b \pmod{m} \leftrightarrow a \pmod{m} = b \pmod{m}$   
证明过程: 当  $a \pmod{m} = b \pmod{m}$  时, 假设  $a \pmod{m} = b \pmod{m} = n$ , 其中  $n \in \mathbb{Z}$  且  $0 \leq n < m$   
则  $\exists s, t \in \mathbb{Z} (a = n + ms \wedge b = n + mt)$ , 于是有  $a - b = m(s - t)$   
又  $s - t \in \mathbb{Z}$ , 则  $m | (a - b)$ , 即  $a \equiv b \pmod{m}$   
当  $a \equiv b \pmod{m}$  时, 有  $m | (a - b)$ , 即  $\exists s \in \mathbb{Z} (a - b = ms)$  即  $a = b + ms$   
假设  $b \pmod{m} = n$ , 即  $\exists t \in \mathbb{Z} (b = n + mt)$ , 其中  $n \in \mathbb{Z}$  且  $0 \leq n < m$   
则  $a = n + mt + ms = n + m(t + s)$ , 又  $t + s \in \mathbb{Z}$ , 则  $a \pmod{m} = n = b \pmod{m}$   
 $a \equiv b \pmod{m}$  当且仅当 存在整数  $k$  使得  $a = b + km$   
即有  $a \equiv b \pmod{m} \leftrightarrow \exists k \in \mathbb{Z} (a = b + km)$

同余类 (congruence class, 或 residue class), 指对于整数  $a$  和正整数  $m$ , 所有和  $a$  模  $m$  同余的整数集合  
有记号表示  $a$  模  $m$  的同余类为  $\bar{a}_m$ , 如通过上下文可确定  $m$ , 也记作  $[a]$   
即有  $a$  模  $m$  的同余类  $\bar{a}_m = \{x \in \mathbb{Z} \mid x \equiv a \pmod{m}\} = \{a + km \mid k \in \mathbb{Z}\}$   
其中每个元素都可作为同余类的代表, 称为代表数 (representative)  
通常可用同余类中的元素  $r$ , 有  $0 \leq r < m$ , 来作为代表数

保同余 指如果  $a \equiv b \pmod{m}$  且  $c \equiv d \pmod{m}$ , 其中  $m \in \mathbb{Z}^+$ , 则有  
 $(a+c) \equiv (b+d) \pmod{m}$  (加法) 且  $ac \equiv bd \pmod{m}$   
即有  $(a \equiv b \pmod{m}) \wedge (c \equiv d \pmod{m}) \rightarrow ((a+c) \equiv (b+d) \pmod{m}) \wedge (ac \equiv bd \pmod{m})$   
注意:  $(ac \equiv bd \pmod{m}) \wedge (c \equiv d \pmod{m}) \rightarrow a \equiv b \pmod{m}$  不成立  
取  $m | c$  且  $m | d$ , 则  $\forall a, b \in \mathbb{Z} (ac \equiv bd \pmod{m})$   
 $(a \equiv b \pmod{m}) \wedge (c \equiv d \pmod{m}) \rightarrow a^c \equiv b^d \pmod{m}$  也不成立  
取  $1 \equiv 4 \pmod{3}, 5 \equiv 2 \pmod{3}, 5^1 \pmod{3} = 2$  而  $2^4 \pmod{3} = 1$   
由此推出在计算机中常用的降低运算量的方法  
即  $(a+b) \pmod{m} = ((a \pmod{m}) + (b \pmod{m})) \pmod{m}$   
 $ab \pmod{m} = ((a \pmod{m})(b \pmod{m})) \pmod{m}$

后者在进行大整数乘积求模运算中可以有效降低运算规模

# Discrete

## Mathematics - P42

mod 含义

注意记号 mod 在  $a \equiv b \pmod{m}$  和  $a \bmod m = b$  中的含义不同

在  $a \equiv b \pmod{m}$  中, mod 可视为标记符 (identifier), 用于表示两个整数间的关系

在  $a \bmod m = b$  中, mod 可视为运算符 (operator), 用于表示求模运算

所以  $\bmod m$  也可视为一个函数, 即  $\bmod(m)(a) = a \bmod m$ , 其中  $a \in \mathbb{Z}$

如在 Haskell 中有:  $t \in \text{mod} \rightarrow (\text{mod} \rightarrow \text{Integral}) :: (\text{Integral} a) \Rightarrow a \rightarrow a$

模算术

(modular arithmetic) 指以一个整数  $m \geq 2$  为模数所做的计算

完全剩余系

(complete residue system), 指包含  $n$  个整数 ( $m \geq 2$ ) 的集合, 任意两个元素模  $n$  不同余

any set of  $m$  integers, no two of which are congruent mod  $m$

即有对于  $\{a_1, a_2, \dots, a_m\}$  ( $a_1, a_2, \dots, a_m \in \mathbb{Z}, m \geq 2$ ), 有  $\forall i, j$  ( $a_i \neq a_j \rightarrow a_i \not\equiv a_j \pmod{m}$ )

最小剩余系 (least residue system modulo  $m$ ), 指对于整数  $m \geq 2$ , 集合  $\{0, 1, \dots, m-1\}$

记为  $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$

模  $m$  算术

指定  $\mathbb{Z}_m$  上的算术运算

模  $m$  加法 用  $+_m$  表示  $a +_m b = ((a \bmod m) + (b \bmod m)) \bmod m$  [注意等式右侧为

模  $m$  乘法 用  $\cdot_m$  表示  $a \cdot_m b = ((a \bmod m) \cdot (b \bmod m)) \bmod m$  ] 普通的整数加法和乘法

运算  $+_m$  和  $\cdot_m$  满足普通加法和乘法的许多同样的性质

模  $m$  运算:  $+_m, \cdot_m$

普通实数运算: +, ·

定义域

$m$  的最小剩余系  $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$

实数集  $\mathbb{R}$

封闭性

$\forall a, b \in \mathbb{Z}_m \wedge a \cdot_m b \in \mathbb{Z}_m$

$\forall a, b \in \mathbb{R} \wedge a \cdot b \in \mathbb{R}$

结合律

$\forall a, b, c \in \mathbb{Z}_m \wedge b \in \mathbb{Z}_m \wedge c \in \mathbb{Z}_m \rightarrow$

$\forall a, b, c \in \mathbb{R} \wedge b \in \mathbb{R} \wedge c \in \mathbb{R} \rightarrow$

$$(a +_m b) +_m c = a +_m (b +_m c) \wedge$$

$$(a + b) + c = a + (b + c) \wedge$$

$$(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

交换律

$\forall a, b \in \mathbb{Z}_m \wedge b \in \mathbb{Z}_m \rightarrow a +_m b = b +_m a \wedge$

$\forall a, b \in \mathbb{R} \wedge b \in \mathbb{R} \rightarrow a + b = b + a \wedge$

$$a \cdot_m b = b \cdot_m a$$

$$a \cdot b = b \cdot a$$

单位元

0 和 1 分别是  $+_m$  和  $\cdot_m$  的单位元

0 和 1 分别是 + 和 · 的单位元

$\forall a \in \mathbb{Z}_m \rightarrow a +_m 0 = 0 +_m a = a \wedge a \cdot_m 1 = 1 \cdot_m a = a$

$\forall a \in \mathbb{R} \rightarrow a + 0 = 0 + a = a \wedge a \cdot 1 = 1 \cdot a = a$

加法逆元

$\forall a \in \mathbb{Z}_m \wedge a \neq 0 \rightarrow m - a$  为  $a$  的加法逆元

$\forall a \in \mathbb{R} \rightarrow$  存在一个  $a$  的加法逆元  $-a \in \mathbb{R}$

$$\exists a \in \mathbb{R} \wedge a + (-a) = 0$$

分配律

$\forall a, b, c \in \mathbb{Z}_m \wedge b \in \mathbb{Z}_m \wedge c \in \mathbb{Z}_m \rightarrow$

$\forall a, b, c \in \mathbb{R} \wedge b \in \mathbb{R} \wedge c \in \mathbb{R} \rightarrow$

$$a \cdot_m (b +_m c) = (a \cdot_m b) +_m (a \cdot_m c) \wedge$$

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \wedge$$

$$(a +_m b) \cdot_m c = (a \cdot_m c) +_m (b \cdot_m c)$$

$$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$$

注意模  $m$  运算并不满足普通实数乘法的逆律, 即不保证有乘法逆元

# Discrete

## Mathematics - P43

交换群 (commutative group) 也称阿贝尔群 (abelian group), obey the axiom of commutativity

指满足其元素运算不依赖于它们的次序 (交换律公理) 的群

如  $\mathbb{Z}_m$  连同模  $m$  加法构成一个交换群

交换环

交换环 (commutative ring), 指乘法满足交换律的环, in which the multiplication operation is

如  $\mathbb{Z}_m$  连同模  $m$  加法和乘法构成一个交换环

commutative

如果  $n, m$  是大于 1 的整数且  $n|m$ , 并且如果  $a \equiv b \pmod{m}$ , 则有  $a \equiv b \pmod{n}$

即有当  $n, m \in \mathbb{Z} \wedge n, m > 1 \wedge n|m$ , 有  $\forall a, b \in \mathbb{Z} (a \equiv b \pmod{m} \rightarrow a \equiv b \pmod{n})$

证明如设  $a = b + ms, m = kn$ , 其中  $s, k \in \mathbb{Z}$ , 且  $k > 0$ , 则有  $a = b + nks$

如果对于整数  $a, b, c > 0, m \geq 2$ , 并且  $(a \equiv b \pmod{m})$ , 则有  $ac \equiv bc \pmod{mc}$

即有当  $c, m \in \mathbb{Z} \wedge c > 0 \wedge m \geq 2$ , 有  $\forall a, b \in \mathbb{Z} (a \equiv b \pmod{m} \rightarrow ac \equiv bc \pmod{mc})$

证明如令  $a = b + ms$ , 则有  $ac = bc + (mc)s$ , 又  $mc \geq 2$ , 则有  $ac \equiv bc \pmod{mc}$

如果对于整数  $a, b, k \geq 1, m \geq 2$ , 并且  $a \equiv b \pmod{m}$ , 则有  $a^k \equiv b^k \pmod{m}$

即当  $k, m \in \mathbb{Z} \wedge k \geq 1 \wedge m \geq 2$ , 有  $\forall a, b \in \mathbb{Z} (a \equiv b \pmod{m} \rightarrow a^k \equiv b^k \pmod{m})$

证明如当  $a \equiv b \pmod{m}$ , 则有  $a \pmod{m} = b \pmod{m}$

于是  $a^k \pmod{m} = (\overbrace{a \cdot a \cdot \dots \cdot a}^k \pmod{m}) = (\overbrace{(a \pmod{m}) \cdot (a \pmod{m}) \cdot \dots \cdot (a \pmod{m})}^k) \pmod{m}$

$= ((b \pmod{m}) \cdot (b \pmod{m}) \cdot \dots \cdot (b \pmod{m})) \pmod{m} = (\overbrace{b \cdot b \cdot \dots \cdot b}^k) \pmod{m}$

$= b^k \pmod{m}$  即有  $a^k \equiv b^k \pmod{m}$

$n$  的  $b$  进制展开, 指可用  $(a_k a_{k-1} \dots a_1 a_0)_b$  来唯一地表示正整数  $n$ , 其中  $b$  为大于 1 的整数,  $k$  为非负整数,  $a_k, \dots, a_0$  为小于  $b$  的

即 对于大于 1 的整数  $b$ ,  $\forall n \in \mathbb{Z} \exists! (a_k, a_{k-1}, \dots, a_0) (k \in \mathbb{N} \wedge a_k, a_{k-1}, \dots, a_0 \in \{0, 1, \dots, b-1\} (\text{即 } P_b) \wedge$

$n = a_k b^k + a_{k-1} b^{k-1} + \dots + a_1 b + a_0 \wedge a_k \neq 0)$

证明存在: 基础步(归纳): 当  $n=1$  时,  $n=1 \cdot b^0$ , 此时  $k=0, a_0=1, a_k \neq 0$  为真

(已证归纳法) 递归步(归纳): 假设当  $n \leq b$  时,  $\forall 1 \leq j \leq n$  可被唯一地表示成  $(a_k a_{k-1} \dots a_0)_b$ , 即  $P_0 \wedge \dots \wedge P_n$

则对于  $P(n+1)$ , 存在唯一的  $0 \leq r < b$  和  $d$ , 使得  $n+1 = db+r$

由于  $n+1 > b$ , 则有  $1 \leq d \leq n$ , 于是  $P(d)$  为真, 又  $r \in \{0, 1, \dots, b-1\}$

则  $n+1 = b(a_k b^k + \dots + a_0) + r = a_k b^{k+1} + \dots + a_0 b + r$ , 故  $P(n+1)$  为真

证明唯一: 假设  $n$  有两种表示且其中  $a_i \neq a'_i, a_j \neq a'_j$  且  $i > j$

则  $a_i b^i + a_j b^j = a'_i b^i + a'_j b^j$ ,  $b^{i-j}(a_i - a'_i) = a'_j - a_j$ , 又  $i-j \geq 1$

$b^{i-j}|a_i - a'_i| \geq b$ , 又  $a_j, a'_j \in \{0, 1, \dots, b-1\}$ , 则  $|a'_j - a_j| < b$ , 存在矛盾

所以  $n$  有唯一的  $b$  进制展开

# Discrete

## Mathematics - P44

构造 b 进制展开

procedure base  $b$  expansion ( $n, b \in \mathbb{Z}^+, b > 1$ )

```

 $q := n$             $8A = 0 \quad 1000 = 0 \quad 1100 = 0 \quad n = 0$ 
 $k := 0 \quad (a_k := 0) \quad a_0 = q \bmod b \quad | \quad q := n$             $b \dots + a_0$ 
while  $q \neq 0$        $a_1 := q \bmod b \quad | \quad q := q \bmod b$             $2 \quad b(b \dots + a_1) + a_0$ 
 $a_k := q \bmod b \quad | \quad a_2 := q \bmod b \quad | \quad q := q \bmod b$             $3 \quad b(b(b \dots + a_2) + a_1) + a_0$ 
 $q := q \bmod b$             $\vdots$             $\vdots$ 
 $k := k + 1 \quad | \quad a_{k-1} := q \bmod b \quad | \quad q := q \bmod b$             $k = b(\dots b(b(c_0) + a_{k-1}) + a_{k-2}) + a_0$ 
return  $(a_{k-1} a_{k-2} \dots a_0)_b$  {n的b进制展开式}

```

注意：在循环前将  $a_0$  初始化为 0，可以覆盖  $n=0$  的情况，即  $n \in \mathbb{N}$

在循环前取  $q=1$ ，并根据  $n$  的正负号设置 signed，并在返回前添加到结果，可覆盖  $n \in \mathbb{Z}$  的情况

另外注意：这个算法可看作霍纳法(Horner's method)的逆运算逻辑。

特别的有，构造一个正整数  $n$  的  $b$  进制展开式可看作一个找零的贪婪算法。

procedure base  $b$  expansion ( $n, b \in \mathbb{Z}^+, b > 1$ )

```

找到 k < n  $k := 0$             $|$  找到  $k \in \mathbb{N}$             $n =$ 
while  $b^k \leq n$             $|$  使得  $a_k = n \cdot b^k + rem$ 
 $k := k + 1$             $|$   $b^k \leq n < b^{k+1} \quad | \quad a_k = n \bmod b^k + a_{k-1} \cdot b^{k-1} + rem$ 
 $|$   $k := k - 1$             $|$  每一步尽可能
 $|$  while  $k \geq 0$             $|$  多地拿面值
 $| \quad a_k := n \bmod b^k$             $|$   $a_k b^k + a_{k-1} b^{k-1} + \dots + a_0 + 0$ 
 $| \quad n := n \bmod b^k$             $|$   $C_k = b^k$  的面值
 $| \quad k := k - 1$             $|$   $c_k = b^k$  的面值

```

return  $(a_k a_{k-1} \dots a_0)_b$  {n的b进制展开式}

注意：关于输出的正确性检查需要注意展开式的首位必须不等于 0（ $n$  为正整数时）

对于第一种算法，可见最后一个生成的是  $a_{k-1}$ ，即需要证明  $a_{k-1} \neq 0$ 。

当循环结束时  $q_k = 0$ ，而上一循环未结束，则  $q_{k-1} > 0$ 。

又  $q_{k-1} = b \cdot q_k + a_{k-1}$ ，可见  $a_{k-1} = q_{k-1} > 0$ 。

对于第二种算法， $a_k$  是循环第一次生成的。

又在循环开始时，有  $b^k \leq q < b^{k+1}$ ，于是有  $1 \leq q/b^k < b$ ，即  $a_k \geq 1 > 0$ 。

于是可知生成的  $n$  的  $b$  进制展开式是合法输出。

常用

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
十进制(decimal)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
二进制(binary)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
八进制(octonary)	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
十六进制(hexadecimal)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

# Discrete

## Mathematics - P45

二进制加法算法

二进制加法算法

procedure add (正整数  $(a_{n-1}a_{n-2}\dots a_0)_2, (b_{n-1}b_{n-2}\dots b_0)_2$ )

$C := 0$

for  $j := 0$  to  $n-1$

$d := L(a_j + b_j + C) / 2^j$

$s_j := a_j + b_j + C - 2d$

$C := d$

$S_n := C$

return  $(S_n S_{n-1} \dots S_0)_2$

】 初始化进位  $C$  为 0

也可以选择直接初始化  $(S_n S_{n-1} \dots S_0)_2$

】 对应相应位的  $a_j$  和  $b_j$

for  $j := 0$  to  $n-1$

取  $d = (a_j + b_j + C) \text{ div } 2^j$

$s_j = (a_j + b_j + C) \bmod 2^j$

更新  $C$  的值

$S_{j+1} := L(a_j + b_j + s_j) / 2^{j+1}$

$S_j := a_j + b_j + s_j - 2s_{j+1}$

】 和的最高位为保留的  $C$

注意此处并未检查最高位是否为 0

加法算法的循环不变量：

断言：在第  $j+1$  次迭代（循环计数器  $0 \leq j < n$ ）开始前， $(S_{j-1} \dots S_0)_2$  是对  $(b_{j-1} \dots b_0)_2$  的求和， $C$  为求和的进位

初始化：在第 1 次迭代开始前（循环计数器  $j=0$ ）， $(S_{j-1} \dots S_0)_2$  为零，进位  $C=0$  (高  $j-1$  位)

保持：在第  $j+1$  次迭代开始前（循环计数器  $j$ ）， $(S_{j-1} \dots S_0)_2$  为已求和部分，进位  $C$  保留已求和的进位

迭代中  $S_j = (a_j + b_j + C) \bmod 2^j$ ,  $C = (a_j + b_j + C) \text{ div } 2^j$

于是第  $j+2$  次迭代开始前（循环计数器  $j+1$ ）， $(S_j \dots S_0)_2$  为已求和部分，进位  $C$  保留到高位求和

终止：终止时循环计数器  $j=n$ ，则此时  $(S_{n-1} \dots S_0)_2$  已完成求和，由于没有  $a_n, b_n$ ，则直接取  $S_n = C$

二进制乘法算法

procedure multiply (正整数  $(a_{n-1}a_{n-2}\dots a_0)_2, (b_{n-1}b_{n-2}\dots b_0)_2$ )

for  $j := 0$  to  $n-1$

if  $b_j = 1$  then  $c_j := a \ll j$

else  $c_j := 0$

$ab = acb_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \dots + b_02^0$

$c_j = a \cdot b_j \cdot 2^j$ ,  $b_j = 0$  且  $j=0$

$b_j = 1$  且  $c_j$  为  $a$  的第  $j$  位

$P := 0$

for  $j := 0$  to  $n-1$

$P := P + c_j$

】 求和部分乘积

$P = \sum_{j=0}^{n-1} c_j \times b_j = ab$

return  $P$

注意这个算法需要  $O(n^2)$  次移位，需要  $O(n^2)$  次二进制位加法

注意如果采用不同的拆解方法，即  $ab = a(c(b_{n-1}^2 \dots (b_0^2 + b_{n-1}) \dots + b_0)) = 2((2 \dots (2(b_0 + ab_{n-1}) \dots + b_{n-1}) + b_{n-2}) \dots + b_0)$

$P := 0$

for  $j := n-1$  down to 0

$P := P * 2$

$P := P + a \cdot b_j$

】 循环不变量：在第  $k$  次迭代开始前， $P = a \cdot cb \text{ div } 2^{n-k}$

初始化：第一次迭代开始前，循环计数器为  $j=n-1$

由于  $b < 2^n$ , 所以  $P = Q \cdot (cb \text{ div } 2^n) = 0$  为真

保持：第  $k$  次迭代开始前，循环计数器  $j=n-k$ ,  $P = a \cdot cb \text{ div } 2^{n-k}$

$P = (P \cdot 2^k) + a \cdot b_j$

$P = (Q \cdot (cb \text{ div } 2^{n-k})) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$

$P = Q \cdot (cb \text{ div } 2^{n-k}) \cdot 2^k + a \cdot b_j$