

Illusions of illustrations · zodiac

2016.11.20~2017.1.8

目录

一.“圆向量”	3
I.引入	3
1.定义	3
2.规则	4
II.最小公倍数相关	4
III.最大公因数相关.....	5
IV.圆向量的起源	8
V.一次不定方程.....	9
1.二元一次不定方程.....	9
2.多元一次不定方程及“超级辗转相除法”	10
3.程序：找一组 n 元一次不定方程的整数解	15
4.程序：给定系数的各阶多元一次不定方程的通解	26
二.多项式定理的计组合数问题	45
1.程序：多项式定理的组合数问题的所有解	46
三.Faà di Bruno's Formula 之遍历 m_i 的组合方式	49
1.程序：找各 m_i 组合的非人工手段	49
2.程序：各项系数均正的多元一次不定方程的非负整数解	52

四.Zengrams:幻觉艺术..... 59

I.引入	59
II.理论解释(part 01).....	60
III.理论解释(part 02).....	63
IV.理论解释(part 03)	66
V.理论解释(part 04)	68
1.程序: Zengrams 幻觉艺术.....	69

五.二元一次不定方程的两个奇异解之间的最小整数解..... 74

I.引入	74
II.用圆向量求最小整数解	75
1.程序: 二元一次方程的奇异解	81
III.上一个理论的分身: 另一个镜像理论	89
1.程序: 二元一次方程的奇异解—镜像.....	90

六.多个理论和对应的程序的内容及结构的优化 99

I.对第一个程序(多元一次不定方程的特解之特殊解法)的改良.....	99
1.程序: 第 1 个程序的改良版(1+5)	99
II.对第二个程序(多元一次不定方程的通解之一般解法)的改良	113
1.程序: 第 2 个程序的改良版(2+5)	113
III.对第之前的第 II.个程序(或者说是第 10 个程序)的再次改良.....	134
1.程序: 第 10 个程序的改良版(2+5+8a)	135
IV.对第之前的第 II.个程序(或者说是第 10 个程序)的另一种方向上的改良.....	170
1.程序: 第 10 个程序的改良版(2+5+8b)	170




后记..... 205

一. “圆向量”

An interesting theory(or just a model) created by the past one of me:

I.引入

1.定义

$a/b=c\dots d$ 对应的 $a=i*b+d$ 中【注：以上和以下的 $a=i*b+d$ 中的 i 和 d 并非一定等于 $a/b=c\dots d$ 中的 c 和 d ，而是 $i\in[1,c]$ 】，记其中： a = 一个周长为 a 的大圆 ，上面有 a 个间隔为单位一的刻度(没画出来)，即 $a=1*a=a*1$ ，并且规定这个大圆也相当于一个步长为 a 的步子 ，它有方向，规定其方向为： $a>0$ 对应于顺时针的步长为 a 的一个走一步便走到出发点的步子； b = 一个步长为 b 的步子  [一段在圆上有方向且长度为 b 的弧线段]，上面有 b 个间隔为单位一的刻度(没画出来)，即 $b=1*b=b*1$ ；并且规定以步长为 b 的步子(以后取名为步子 b)的起点和步子 a 的起点相同，且同样地依照 a 的标准，规定 $b>0$ 对应于步子 b 以步长 b 在圆上顺时针地迈(事实上我们经常先/直接就规定步子 b 的起点和方向，让步子 a 来迁就和同步于步子 b ，或者有时候甚至忘了 a 是个步子/向量)； i = 步子 b 在以步长 b 跳跃时的总步数 i ，且规定 $i>0$ 时以步子 b 的方向迈， $i<0$ 时向步子 b 的反方向迈； d = 向量 $a-i*$ 向量 b 。【注：圆向量允许 a 比 b 小(大圆比步子的步长小)，且允许任意字母为负，但不常这样做。】

【现在来看，圆向量所研究的内容，更准确地说是， $ax-by$ 的值的分布情况：相对于某(些)个 $ax=0$ 的(在圆上 or 在数轴上的)相对位置；以上内容相当于引入的是 $x=1$ 时候的事情： $a-i*b=d$ 的分布情况；而接下来，“ x 的可变”便将成为舞台的主角。】

即以上 a 、 b 、 i 、 d 中，只有 i 是标量，而 a 、 b 、 d 均为(伪)向量(步子=向量，步长=向量的模长)，且均为一维向量，相当于向上凸起后卷曲至首尾相接的一段 x 轴上的向量，只有正负两个方向，分别代表之后的顺时针和逆时针方向。

2.规则

圆向量有两个十分显著的特点: **rule1**.由于 $i*b=a-d=[(-d)+a]\sim(-d)$, 所以 $i*b\sim(-d)$, 其中 “ \sim ” 表示 “等价于”。因而接着有: $k*a=k(i*b+d)=ki*b+k*d=ki*b+[i'*b+d']=[ki+i']*b+d'=i''*b+d'$, 因而有 $i''*b=k*a-d'=[(-d')+k*a]\sim(-d')$, 且有 $i''*b=[(a-d')+(k-1)*a]\sim(a-d')$, 即有 $i''*b\sim(a-d')\sim(-d')$ 。**rule2**.如果 $i*b-a=-d$, 那么 $2i*b-i*b=(a-d)\sim(-d)$; 即若有 $k*a-d=i*b$, 那么 $k*a-nd\sim(a-nd)+(n-1)a=n(a-d)\sim n(k*a-d)=ni*b$; 更一般地有: $i1*b+d=i2*b$, 那么 $i1*b+nd=ni2*b-(n-1)i1*b=[ni2-(n-1)i1]*b$, 那么 $i1*b+(d-k*a)\sim i2*b$ 且 $i1*b+n(d-k*a)\sim[ni2-(n-1)i1]*b$, 若令 $(d-k*a)=d'$, 那么上述表述将变为: 若有 $i1*b+d'\sim i2*b$ 那么有 $i1*b+nd'\sim[ni2-(n-1)i1]*b$ 。【若令 $i1*b=k0*a$ (总有一对 $k0$ 和 i 满足此等式), 抑或者令 $i1*b+d'\sim i2*b$ 中 $i1=0$ 并在左边加上 $k*a$, 那么就有之前的 $k*a-nd\sim ni*b$ 。】进一步地, 若有 $i1*b+d=i2*b$, 则有 $i3*b+nd=[i3+n(i2-i1)]*b$, 设 d 变为 $d'=d-k*a$, 上式等号会改为等价于, 再令 $d'=d$ 则有: 若 $i1*b+d\sim i2*b$, 则 $i3*b+nd\sim[i3+n(i2-i1)]*b$ 。

在创造圆向量来研究各种各样的 i 下的 $i*b$ 步子在以 a 为周长的圆上留下的脚印分布情况时, **Rule1**.昭示了: $i*b$ 等效于 $i*b\pm k*a$; **Rule2**.昭示了: 进一步利用 rule1, 我们便有: 如若有一个 d 被 found 使得 $i1*b+d\sim i2*b$ 成立, 那么 $i1*b+nd\sim[ni2-(n-1)i1]*b$ 将恒成立。这句话什么意思呢: 对于任意步数 $=i1$ 所对应的圆 a 上的某点 $i1*b$, 如果发现以此 $i1*b$ 向量的终点为起点的一个 d 向量的终点, 与原点为起点所构成的向量, 的模长仍是步长 b 的整数倍, 即属于步子 b 在圆 a 上的落脚点之一, 那么以此 d 向量的终点为起点, 再画任意 i 个 d 向量后, 其终点所在的点, 一定仍然是步子 b 在圆 a 上的落脚点之一。进一步地, 若 $i1*b+d\sim i2*b$, 则 $i3*b+nd\sim[i3+n(i2-i1)]*b$ 。这句话便更加地将原 rule2 拓宽至: 任意一个步子 b 的落脚点 $i2*b$, 相对于任意另一个步子 b 的落脚点 $i1*b$ 的向量 d , 若以任意一个步子 b 的落脚点 $i3*b$ 为它的起点, 那么它的终点一定是步子 b 的落脚点之一。

II.最小公倍数相关

(1).下面我们来充分利用 rule1, 求 $[a,b]$: 不妨通过调换顺序, 使得 $a\geq b$, 此时便有 $[a,b]=\{\frac{a-b}{(a-b,b)}\cdot 1+\frac{b}{(a-b,b)}\cdot 1\}\cdot b=\{\frac{a-2\cdot b}{(a-2\cdot b,b)}\cdot 1+\frac{b}{(a-2\cdot b,b)}\cdot 2\}\cdot b=\dots=\{\frac{a-c\cdot b}{(a-c\cdot b,b)}\cdot 1+\frac{b}{(a-c\cdot b,b)}\cdot c\}\cdot b$, 这是什么意思呢: 假如有 $a=i\cdot b+d$, (其中 $i=1\sim c$) 那么 $a-i\cdot b$ 表示步子 b 走了 i 步后向量 $i*b$ 的等价向量 $(-d)$ 的模长: $|-d|=-(-d)=d$, 而走了一步步子 b 的模长为 b , 则能使 $k1*(-d)+k2*b=0$ 成立的那对最小的 $(k1,k2)$, 即为能使步子 b 走

k_1*i+k_2*1 步后第一次回到出发点(原点)【至于为什么是第一次回到出发点, 后面才能知道。。】的那对 (k_1,k_2) , 即为 $(k_1*i+k_2*1)*b$ 后将得到 a 、 b 的最小公倍数 $[a,b]$ 的那对 (k_1,k_2) 。而使 $k_2*b=k_1*d=k_1*(a-i*b)$ 成立的最小的一对 (k_1,k_2) 必能使 $k_2*b=k_1*(a-i*b)=[a-i*b,b]$, 且能使 $(a-i*b)/k_2=b/k_1=(a-i*b,b)$ 。【这里其实我们已经推知了最大公约和最小公倍的性质: $a*b=[a,b]*(a,b)$; 具体的推导过程如下所示: 要使

①. $k_2*b=k_1*a=K$ 且 K 在各对 k_1,k_2 的组合中最小, 此时由于 $a|K$ 、 $b|K$ 且 K 最小, 那么定义 or 记 $K=[a,b]$ 。②.由于 k_1,k_2 要使 K 最小, 那么 k_1 、 k_2 所共有的所有因数之积必须最小(否则 K 不是最小), 而最小是 1, 即 k_1 、 k_2 的最大公因数为 1, 记为 $(k_1,k_2)=1$, 并称 k_1 、 k_2 互质。③.又由于 $k_2/k_1=a/b$, 所以 $k_2/k_1=a/b=[a/(a,b)]/[b/(a,b)]$, 即当 $k_2=a/(a,b)$ 且 $k_1=b/(a,b)$ 之时, 才能同时满足 $k_2/k_1=a/b$ 且 $(k_2,k_1)=(a/(a,b),b/(a,b))=(a,b)/(a,b)=1$ 】那么由以上两式子便可得出 $k_1=[a-i*b,b]/(a-i*b)=b/(a-i*b,b)$, $k_2=[a-i*b,b]/b=(a-i*b)/(a-i*b,b)$, 带入 “ $\{k_1*i+k_2*1\}$ 步* b 步长/步” 便有 $[a,b]=\{\frac{a-i*b}{(a-i*b,b)} \cdot 1 + \frac{b}{(a-i*b,b)} \cdot i\} \cdot b = \{\frac{[a-i*b,b]}{b} \cdot 1 + \frac{[a-i*b,b]}{a-i*b} \cdot i\} \cdot b$ 。

现把用最大公因数表示 $[a,b]$ 的, 能直接通分的式子通分: $[a,b]=\{\frac{a-i*b}{(a-i*b,b)} \cdot 1 + \frac{b}{(a-i*b,b)} \cdot i\} \cdot b = \frac{a*b}{(a-i*b,b)}$ 。其中 $[a,b]=\frac{a*b}{(a-i*b,b)}$ 加上我们在之前就已经得到过的: 在讨论 k_1 、 k_2 在使 $k_2*b=k_1*(a-i*b)$ 成立时为怎样的最小的一对的时候: $[a,b]=\frac{a*b}{(a,b)}$, 则有 $\frac{a*b}{(a,b)} = \frac{a*b}{(a-i*b,b)}$, 便有 $(a,b)=(a-i*b,b)=(b,a-i*b)=(a-i*b,b-i'*(a-i*b))=(b-i'*(a-i*b), (a-i*b)-i'[b-i'*(a-i*b)])$ ……于是我们可以先以这种辗转相除法【由于这里 i 不一定等于 c , $i \in [1,c]$, 所以也可叫做“更相减损术”】算出 (a,b) , 再给出 $[a,b]$ 。另一方面, 如若强制不让最大公因数参与运算, 那么便只能用 $[a,b]=\{\frac{[a-i*b,b]}{b} \cdot 1 + \frac{[a-i*b,b]}{a-i*b} \cdot i\} \cdot b$, 然后再以同样的方式求 $[a-i*b,b]=[b,a-i*b]$ 至可算的末层 or 可算的某层; 或者作弊: $\frac{a*b[a-i*b,b]}{(a-i*b)*b}$ 。

(2).另一种用圆向量解释的, 除了“余数”还利用了“剩数”的, 求 $[a,b]$ 的方法: $[a,b]=\{\frac{(c+1)*b-a}{((c+1)*b-a,a-c*b)} \cdot c + \frac{a-c*b}{((c+1)*b-a,a-c*b)} \cdot (c+1)\} \cdot b$, 这是什么意思呢: 假设有 $a=c*b+d$, 则 $c*b=a-d \sim (-d)$, 且 $(c+1)*b=a+b-d \sim b-d$, 那么能使 $k_1*(-d)+k_2*(b-d)=0$ 的最小一对 k_1 、 k_2 , 即为能使 $\{k_1*c+k_2*(c+1)\} \cdot b$ 为 $[a,b]$ 的那对 k_1 、 k_2 。如果我们进一步化简 $[a,b]=\{\frac{(c+1)*b-a}{((c+1)*b-a,a-c*b)} \cdot c + \frac{a-c*b}{((c+1)*b-a,a-c*b)} \cdot (c+1)\} \cdot b = \frac{a*b}{(b-d,d)}$ 。可以发现这个结果 $\frac{a*b}{(b-d,d)}$ 是 $\frac{a*b}{(a-i*b,b)}$ 的升级版: 令 $i=c$ 则 $\frac{a*b}{(a-i*b,b)} = \frac{a*b}{(a-c*b,b)} = \frac{a*b}{(d,b)} = \frac{a*b}{(b-d,d)}$ 。

III.最大公因数相关

接下来我们顺水推舟: 进一步地开始系统地研究圆向量所想研究的问题: 各种各样的 i 下的 $i*$ 步子 b 在以 a 为周长的圆上留下的脚印分布情况; 特别是脚印是否分布均匀; 如果否, 最小间距是多少, 如果是, 是否遍及所有整数刻度; 如果是, 什么情

况下是，如果否，那么间隔是多少 and 怎么算 and 有何物理意义。并给出(a,b)的具体求法及这种求法的得来过程，这是一种类似于辗转相除法的方法：

第一个问题：脚印是否分布均匀。这个问题得先解决“什么情况下能足迹遍及所有整数刻度”：假如对于 $k_1 \cdot b = k_2 \cdot a$, $k_1, k_2 \in \mathbb{N}^+$, 且 $(a,b)=1$, 要让 $k_1 \cdot b$ 为 a 的整数倍，则根据以前的讨论，整数 k_1 的最小值为 $a/(a,b)$, 而又由于 $(a,b)=1$, 所以 $k_{1\min}=a$, 即也就是说，步子 b 在周长为 a 的大圆里迈步时，至少得迈 a 次才能第二次落在同一个地点 M , 那么前 $1 \sim a-1$ 次迈步后的落脚点均不在同一个地点，把这个规则用于所有的，路途中的，所迈出了步长为 b 的任意 i 个步子后的落脚点 N_i 们

【 $0 < i < a$ 】，则有： $M = N_0 \neq N_1 \sim a-1$, $N_1 \neq N_2 \sim a$ 则 $N_1 \neq N_2 \sim a-1$, $N_2 \neq N_3 \sim a+1$ 则 $N_2 \neq N_3 \sim a-1$, ..., $N_i \neq N_{i+1} \sim a+i-1$ 则 $N_i \neq N_{i+1} \sim a-1 \dots N_{a-2} \neq N_{a-1} \Rightarrow$ 即有 $N_0 \neq N_1 \sim a-1$ 、 $N_1 \neq N_2 \sim a-1$ 、 $N_2 \neq N_3 \sim a-1$ 、... $N_i \neq N_{i+1} \sim a-1$ 、... $N_{a-2} \neq N_{a-1}$ 。 \Rightarrow 由于 $N_0 \neq N_1 \sim a-1 \Leftrightarrow N_1 \sim a-1 \neq N_0$, 则有 $N_0 \neq N_1 \sim a-1$ 、 $N_i \neq N_{(0 \sim i-1) \cup (i+1 \sim a-1)}$ 【 $i \in [1 \sim a-2]$ 】、 $N_{a-1} \neq N_{[0 \sim a-2]}$ 。即也就是说，这 a 个点 $N_0 \sim N_i \sim N_{a-1}$, 互异。

然而有趣的是，周长为 a 的大圆上的以供步子 b 落脚的互异的整数刻度们【步子 b 的落脚点只能落在整数刻度上，因为 $k_1 \cdot b$ 是整数且规定了起点是整数刻度 0 】，其数量也只有 a 个，那么这 a 个互异的落脚点 $N_0 \sim N_i \sim N_{a-1}$ 只能分别与周长为 a 的大圆上的互异的 a 个整数刻度，一一对应。 \Rightarrow 现又因整数刻度们是均匀分布的，所以脚印是均匀分布的。

已经解决了一个问题：当 $(a,b)=1$ 时，不论步子 a 在 b 圆上走，还是步子 b 在 a 圆上走，落脚点将遍及所有整数刻度，且因此分布均匀。【虽然我们仅仅推知了“落脚点遍及所有整数刻度”的充分条件是 $(a,b)=1$, 其实其必要条件也是它，其得知过程隐藏在下面。】接下来，我们来看一下当 $(a,b) \neq 1$ 时，落脚点的分布情况：是否均匀分布，若仍均匀，相邻刻度之间的间距为多少，怎么算？

对于 $(a',b') \neq 1$, 如果我们沿用以上的模型并对其加以改造：假设有 $a' = (a',b') \cdot a$, $b' = (a',b') \cdot b$, $(a,b)=1$ 。仍然是对于类似的 $k_1 \cdot b' = k_2 \cdot a'$, $k_1, k_2 \in \mathbb{N}^+$, 只不过此时 $(a',b') \neq 1$ 了，那么有 $k_1 \cdot (a',b') \cdot b = k_2 \cdot (a',b') \cdot a$, 即仍有 $k_1 \cdot b = k_2 \cdot a$, 那么我们只需要将之前的大圆 a 的周长乘以 (a',b') 后改大为 a' , 并同时把大圆 a' 的单位一从大圆 a 的相邻刻度之间的间隔 $=1$ 修改为大圆 a' 的相邻刻度的间隔 $= a'/a = (a',b')$, 即把单位间隔 or 单位一的单位扩大/扩大为了 (a',b') , 而刻度总数不变。此时圆 a' 相当于刻度分布、刻度数均没有变化，仅仅刻度与刻度之间的间隔扩大了 (a',b') 的圆 a , 而步子 b 也由于 $\cdot (a',b')$ 变为了 b' 而仍然会总踩在(且踩全)原来的那些刻度上，相当于整体把整个图、动图，都放大了 (a',b') 倍，包括圆 a 和步子 b 。现在再回过头来把圆 a' 的，以 (a',b') 为刻度与刻度之间的间隔的，各刻度之间的，各个单位刻度给补充上，便可以发现这些刻度由于不是 (a',b') 的整数倍而不会成为落脚点。【即由于 $(a',b') \nmid b'$ 且 $(a',b') \nmid a'$, 而

(a',b')不能|这些刻度所对应的向量长度，所以这些刻度所对应的向量长度不能被表示为 $k_1*b'-k_2*a'$ ，即不能成为步子 b' 的落脚点——这才是姗姗来迟的是否能成为落脚点的判据：是否能被表示成 $k_1*b'-k_2*a'$ (虽然之前我们也已经提到过它了)】。

以上已经说明了无论(a,b)是否=1，即对于任意整数 a、b，落脚点的分布均匀，且间隔为(a,b)。如果各位仍尚感这仅仅是“说明”而不是“证明”的话，那么我接下来的风格完全不同的语句就将成为你所想要的另一番风景了【虽然其实上一段我应该用其结尾部分作为开头的，这样再接着铺展开来才会看上去更顺理成章】：

假设脚印分布不一定均匀，那么设相邻落脚点之间有最小间距且为 d_{min} ，下面我们开始一段求 d_{min} 的路途，并最终给出 d_{min} 在数学上对应的物理意义：根据 Rule2：任意一个步子 b 的落脚点 i_2*b ，相对于任意另一个步子 b 的落脚点 i_1*b 的向量 d ，若以任意一个步子 b 的落脚点 i_3*b 为它的起点，那么它的终点一定是步子 b 的落脚点之一。现在我们开始疯狂地挖掘和利用这个规则：为了得到 d_{min} ，我们首先先创造一个模长小于 b 的向量 d ，即有 $a/b=c\cdots d$ 中的余数 d 所对应的向量： $-d$ 【 $c*b=a-d\sim d$ 】，这是 $i\in[1\sim c]$ 时第一次得到的一个绝对值小于 b 的向量： $-d$ ，此时我们再走一步，即令 i (再加 1)= $c+1$ ，得到另一个可能小于 d 的向量 $d'=b+\text{向量 } d=b+(-d)=b-d$ ；此时比较两个向量的模长大小，取 $\min\{d, b-d\}$ 对应的向量 $\{-d, b-d\}$ 中的一个，将它的起点放置在 $\max\{d, b-d\}$ 所对应的向量 $\{-d, b-d\}$ 中的那个向量的终点处，并让它不断地以终点为新起点地自我复制粘贴，所到之处因 rule2 而均为落脚点，即有 $\max\{d, b-d\}/\min\{d, b-d\}=c'\cdots d''$ 中余数 d'' 所对应的向量 d'' ：一个符号/方向与 $\max\{d, b-d\}$ 所对应的向量相同【或者说与 $\min\{d, b-d\}$ 所对应的向量相反】的向量，再走一步，即 $c'+1$ 后，即得到向量 $d'''=\min\{d, b-d\}$ 所对应的向量+向量 d'' ；此时再比较两个向量的模长大小，取 $\min\{|\text{向量 } d''|, |\text{向量 } d''|\}$ 所对应的向量 $\{\text{向量 } d'', \text{向量 } d'''\}$ 中的一个，将它的起点放置在 $\max\{|\text{向量 } d''|, |\text{向量 } d''|\}$ 所对应的向量 $\{\text{向量 } d'', \text{向量 } d'''\}$ 中的那个向量的终点处，并让它不断地以终点为新起点地自我复制粘贴，所到之处仍因 rule2 而均为落脚点，即有 $\max\{|\text{向量 } d''|, |\text{向量 } d''|\}/\min\{|\text{向量 } d''|, |\text{向量 } d''|\}=c''\cdots d''''$ 中余数 d'''' 对应的向量 d'''' ：一个符号/方向与 $\max\{|\text{向量 } d''|, |\text{向量 } d''|\}$ 所对应的向量相同【或者说与 $\min\{|\text{向量 } d''|, |\text{向量 } d''|\}$ 所对应的向量相反】的向量。。。

在无止境的使用这样的规则下去后，到一定程度总会出现 $\max\{|\text{向量 } d_{2n}|, |\text{向量 } d_{2n+1}|\}/\min\{|\text{向量 } d_{2n}|, |\text{向量 } d_{2n+1}|\}=c_n\cdots d_{2n+2}$ 中余数 d_{2n+2} 为 0 的时候，此时 $\min\{|\text{向量 } d_{2n}|, |\text{向量 } d_{2n+1}|\}$ 即为 d_{min} ，而又因此 d_{min} 所对应的向量： $\min\{|\text{向量 } d_{2n}|, |\text{向量 } d_{2n+1}|\}$ 所对应的向量，的起点能被放置于任何已有的落脚点，并对自身进行疯狂繁殖，且这些繁殖后的向量终点由于 rule2 的保证，而一定是步子 b 的落脚点之一，那么这个 d_{min} 连同它的两端的落脚点会遍布且均匀遍布整个大圆 a。那么落脚点会分布均匀且 $d_{min}=(a,b)$ 。

总结一下这个得到 d_{min} 和 (a,b) 的方法: $|向量d_0| = d_0 = \max\{|向量a|, |向量b|\} \% \min\{|向量a|, |向量b|\}$; $|向量d_1| = \min\{|向量a|, |向量b|\} - d_0$; $if(d_0 \neq 0)\{i = -1;$
 $do(i = i + 1; |向量d_{2i+2}| = d_{2i+2} = \max\{|向量d_{2i}|, |向量d_{2i+1}|\} \% \min\{|向量d_{2i}|, |向量d_{2i+1}|\};$
 $|向量d_{2i+3}| = \min\{|向量d_{2i}|, |向量d_{2i+1}|\} - d_{2i+2};); while(d_{2i+2} \neq 0); \}$ 其中 $d_{2n+2} = 0$ 且 $i \in [0 \sim n]$ 。

若把 a 、 b 也看作某种 d_{-2} 和 d_{-1} 的话, 那么再用一维数组的规则将其规范为 $d[0]$ 和 $d[1]$, 就可以进一步简化而写成纯粹的 C 语言便是: $d[0] = a; d[1] = b; i = -1; do\{i = i + 1; d[2i+2] = \max(d[2i], d[2i+1]) \% \min(d[2i], d[2i+1]); d[2i+3] = \min(d[2i], d[2i+1]) - d[2i+2]; \}; while(d[2i+2] \neq 0); d_{min} = (a,b) = d[2i+2];$ 如果不想定义动态数组的话, 上述代码也可以改写并简化为: `#include<stdio.h> int max(int a,int b){if (a>b)return a;else return b;} int min (int a,int b){if (a<b)return a;else return b;} void main(){int a,b,d,e;printf("please input your a,b:");scanf("%d,%d",&a,&b);do{e=min(a,b);d=max(a,b)%e;a=e-d;b=d;}while(d!=0);printf("dmin=(a,b)=%d\n",e);}`

若想用以上代码的话, 注意输入的整数得在 `int` 范围内【不要只想着检验这个算法算的快不快而只顾着刁难它...】且第一行只能为 `#include <stdio.h>`, 之后的代码倒是可以一长串地共处一行。其实以上认识如果用标准的数学语言来描绘的话, 会正如辗转相除法的成立之源一样简单: 由于 $(a,b)|a$, $(a,b)|b$, 所以 $(a,b)|(max(a,b)-c*min(a,b))$, 即 $(a,b)|(max(a,b)\%min(a,b))$, 对于辗转相除法, 它接下来会说: 所以 $(a,b)|(min(a,b)\%[max(a,b)\%min(a,b)])$, 而对于我们的另一种算法, 会说成: 所以 $(a,b)|(max\{min(a,b)-max(a,b)\%min(a,b), max(a,b)\%min(a,b)\}\%min\{min(a,b)-max(a,b)\%min(a,b), max(a,b)\%min(a,b)\})$, 之后每一步, 它们之间的差异也都是这样的。看起来, 还是辗转相除法在运算步骤上更简单, 不管 a 、 b 相差大时还是小时, 因为 $min(a,b)-max(a,b)\%min(a,b)$ 与 $min(a,b)$ 相差并不大且 `ours` 多了个判断大小的步骤。

IV. 圆向量的起源

所以其实一开始我们就该用圆向量来解释辗转相除法的, 而不是另辟此蹊径。所以, 圆向量更像是一种用于把问题抽象化的模型, 或者为了研究问题而开发出来的一种让问题更直观化的方法, 而不一定是最有效的手段。其实我创造它的原因, 很大程度上是起源于这样一个问题: 在一个内壁涂满了银镜的球壳中, 取一个过球心的平面交球得一个内边缘有银镜的环, 现有一个在圆环之内的点, 它不过球心地向着环内壁射出一束在该环所在平面的射线, 射线与内壁触碰后以反射角 θ 反射, 由于反射后还会碰壁, 所以它会一直反射: 这样就会出现两个极端情况: (从垂直于环面的角度看) case1. 激光不会布满环内的面, 而是构成一个有限个角的“规则多角星”, 并正重复地沿着这之前留下的径迹无止境地循环下去; case2. 激光“将会”布满(充斥)环内的

封闭的平面区域(但由于光速有限且光程为 ∞ 而永远不会“布满”所有角落?)。→→→
更精确的说:将是一个总有缝隙,但如果时间允许的话,任何一个指定的缝隙都会被填满的,一个圆环带(由两个同心圆及之间的区域所构成)。

现在我们先尝试着将这个场景转化为圆向量的模型【虽然其实圆向量是来自于它...】:在laser反射的过程中,先任意选取一个已有的反射点所对应的时间点和空间点,再以时间顺序标记出每相隔一个反射点的反射点们在圆环上的空间位置:你会发现,如果仅考虑这些相隔一个反射点的反射点们,它们这些事件在时空中连接起来所构成的一个整体事件,等效于朝向这些反射点们所对应的,剩下的反射点处的光线角度在小于 π 的方向上的变化量 2θ 【if 光线无方向】(或 $\pi-2\theta$ 【if 光线有方向】)所对应的方向/逆方向(或者其逆方向/方向),走一段弧长向量。【而这些剩下的反射点们(它们也是相隔一个反射点的反射点们)的随时间生成的脚印在空间上的印记生成方向,是同种度量标准下的其补集的印记生成方向的反方向】

这样的(相隔一个反射点的)一个个反射点序列就等效于圆向量中的一个一个落脚点序列,因而我们可以用圆向量来解释这个问题: **Case1.的充要条件**是存在 $k_1, k_2 \in \mathbb{Z}$, 使得 $k_1 \cdot b - k_2 \cdot a = 0$, 即存在 $[a, b]$, 使得 $k_1 \cdot b = k_2 \cdot a = [a, b]$ 。这时 $a/b = k_1/k_2 \in \mathbb{Q}$, 即只要 a/b 是个有理数, 那么 $[a, b]$ 存在, 且 case1 成立。而使得 $a/b \in \mathbb{Q}$ 的 a, b , 可以是正负有理数对、相差有理数倍的无理数对等等。 **Case2.的充要条件**是对于任意 $k_1, k_2 \in \mathbb{Z}$, 均有 $k_1 \cdot b - k_2 \cdot a \neq 0$, 即 $k_1 \cdot b$ 所对应的任何一个反射点(落脚点), 过了光线在它那里反射的 moment 之后, 光线无法再次回到此反射点, 该效应所导致的效果便是: 当某时刻, 光线刚从某反射点反射出去后, 它无法落脚于圆上在此时刻之前任意一个已有的反射点(不然对于那一点来说, 就叫“再次回到”), 那么它只能“选取”已有的这些点们在“大圆范围内”(或者说是整条实数轴上)的实数中的补集里的点, 来作为它的落脚点。这所导致的结果便是: 光线的反射点的集合在尝试着穷尽实数范围内的(或者某个实数区间上的)所有点而一直随着时间在扩增元素。而能导致这种局面出现的 a, b , 只能均为相差无理数倍的无理数们。

V.一次不定方程

1.二元一次不定方程

如何判断整数系多项式方程 $ax+by+c=0$ 是否有整数解 (x, y) ? 如果有, 有多少个? 怎么求出其中的某一个? 怎么求其余剩下的解? 这些点在直线 $ax+by+c=0$ 上分布均匀吗? 其中最靠近原点的 (x_0, y_0) 是?

①.对于任意整数 k_1, k_2 , $(a,b)|k_1*b$, $(a,b)|k_2*a$, 则 $(a,b)|(k_1*b+k_2*a)$; 所以如果方程 $ax+by+c=0$ 存在一个整数解 (x_0, y_0) , 则 $(a,b)|(ax_0+by_0)$; 又因 $ax_0+by_0=-c$, 所以 $(a,b)|-c$, 即有: $(a,b)|c$. $\rightarrow\rightarrow\rightarrow$ 于是我们便有: “ $ax+by+c=0$ 有整数解” 是 “ $(a,b)|c$ ” 的充分条件. ②.如果有 $(a,b)|c$, 设 $c=k(a,b)$, $a=k_1(a,b)$, $b=k_2(a,b)$, 其中 $(a,b)=(k_1(a,b), k_2(a,b))=(k_1, k_2)(a,b)$, 即 $(k_1, k_2)=1$, 则 $ax+by+c=0$ 变为 $k_1(a,b)x+k_2(a,b)y+k(a,b)=0$, 即 $k_1x+k_2y+k=0$, 其中 $(k_1, k_2)=1$. 根据以前在圆向量中的讨论, 步子 x 的脚印将布满大圆 y 上的所有整数刻度【或步子 y 的脚印将布满大圆 x 上的所有整数刻度】, 因而 k_1*x+k_2*y 将取遍 $T*y+[0\sim y-1]$ 【或 $T*x+[0\sim x-1]$ 】, 即所有整数, 又因 $k\in\mathbb{Z}$, 所以存在 (x,y) 使得 $k_1x+k_2y+k=0$, 即存在 (x,y) 使得 $ax+by+c=0$. $\rightarrow\rightarrow\rightarrow$ 于是我们便有: “ $ax+by+c=0$ 有整数解” 是 “ $(a,b)|c$ ” 的必要条件.

综上, 对于 $a, b\in\mathbb{Z}$, “ $ax+by+c=0$ 有整数解” 与 “ $(a,b)|c$ ” 互为充要条件.

【注: a, b, c 也可均 $\in\mathbb{Q}$, 以及凡是可以约分约成整数的所有 $a, b, c\in\mathbb{R}$, 即 a, b, c 甚至可以为相差有理数倍的无理数们. 特别地, 可以构造一些含无理数的 a, b, c 组合, 使得直线经过整点, 但只经过一个整点.】并且对于②., 不仅存在 (x_0, y_0) 使得 $ax_0+by_0+c=0$, 而且若设有 $ak_1+bk_2=0$, 其中 $k_1/k_2=-b/a=-[b/(a,b)]/[a/(a,b)]=-[bt/(a,b)]/[at/(a,b)]$, 把两式子相加即有 $a(x_0+k_1)+b(y_0+k_2)+c=0$, 即 $(x,y)=(x_0+k_1, y_0+k_2)=(x_0+t*b/(a,b), y_0-t*a/(a,b))$ 或 $(x_0-t*b/(a,b), y_0+t*a/(a,b))$ 均是 $ax+by+c$ 的解. 由此可见若 $ax+by+c=0$ 有解, 则有无数个解, 且这些解所对应的整点在直线上分布均匀, 且相邻整点距离为 $\frac{(a^2+b^2)^{0.5}}{(a,b)}$.

以上, 如果我们判断出某个方程 $ax+by+c=0$ 有解后, 再寻找到一个解 (x_0, y_0) , 就能确定/求出它的所有剩下的解了. 但是, 问题就在其中: 这个解 (x_0, y_0) 怎么找? ? 我们已经会进行: 1.判断是否有解、2.解的个数、4.已知一个解后的其他所有解, 唯独差的就是: 如何 3.从无到有地求出一个任意解? 为此, 我开发了 a set of rule which is 非常有用的, 它是这么的有用以至于, 在写下这些文字之时, 我仍不知这套规则是怎样 works 的. 下面就引入这样一套规则: 名曰: “超级辗转相除法”. 下面举一个这个规则在二元一次不定方程上的应用: 设有方程 $19x-7y-6=0$, 则 $5x+7(2x-y)-6=0$, 则 $5(3x-y)+2(2x-y)-6=0$, 则 $(3x-y)+2(8x-3y)-6=0$, 则 $(3x-y)+2(8x-3y-3)=0$, 设 $3x-y=0$ 且 $8x-3y-3=0$, 解得 $x=-3$, $y=-9$, 且恰好此解 $(-3, -9)$ 最靠近原点.

2.多元一次不定方程及 “超级辗转相除法”

(1).引入: 判断多元一次方程是否有整数解: “ $ax+by+c=0$ 有整数解” \Leftrightarrow “ $(a,b)|c$ ”, 此前我们已经有过此结论及推导它的过程, 现在让我纯粹地用圆向量解

释它们之间的这个关系，以便扩展下去： $“ax+by+c=0$ 有整数解” \Leftrightarrow “存在步数 $y_0 \in \mathbb{Z}$ ，使得 $(by_0+c)+x*a=0$ ”【或说成存在 y_0 使得 $a|by_0+c$ 】 \Leftrightarrow “存在 $y_0 \in \mathbb{Z}$ ，使得 $by_0+c \sim 0$ ” \Leftrightarrow “存在 $y_0 \in \mathbb{Z}$ ，使得走了 1 步 c 步长的步子 c 后，再走 y_0 步步长为 b 的步子 b 时，能回到起点。” \Leftrightarrow “ $y*b$ 在圆 a 上留下的脚印所对应的向量 $+c$ 向量得到的旋转后的新脚印中，存在一个向量的终点=原点” \Leftrightarrow “脚印 $by+c \sim$ 脚印 $b'y$ ” \Leftrightarrow “ $c=k*d_{min}$ ” $\Leftrightarrow (a,b)|c$ 。

①.由于 $“ax+by+c=0$ 有整数解” $\Leftrightarrow (a,b)|c$ 所以 $“ax+by+cz+d=0$ 有整数解” \Leftrightarrow “存在 z_0 使得 $(a,b)|(cz_0+d)$ ”；又由于 $“ax+by+c=0$ 有整数解” \Leftrightarrow “存在 y_0 使得 $a|by_0+c$ ” 所以 $“存在 z_0 使得 $(a,b)|(cz_0+d)$ ” \Leftrightarrow “ $(a,b)x+cz+d=0$ 有整数解”；又同样由于 $“ax+by+c=0$ 有整数解” $\Leftrightarrow (a,b)|c$ 所以 $“(a,b)x+cz+d=0$ 有整数解” $\Leftrightarrow ((a,b),c)|d$ ”。综上， $“ax+by+cz+d=0$ 有整数解” \Leftrightarrow “存在 z_0 使得 $(a,b)|(cz_0+d)$ ” \Leftrightarrow “ $(a,b)x+cz+d=0$ 有整数解” $\Leftrightarrow ((a,b),c)|d$ ”，又由于 $((a,b),c)|c$ 且 $|a,b|$ 并为满足条件的最大值，则也 $|a$ 且 $|b$ 并为满足条件的最大值，则 $((a,b),c)|c$ 且 $|b$ 且 $|a$ 且为满足条件的最大值，所以 $((a,b),c)=(a,b,c)$ 。则 $“((a,b),c)|d” \Leftrightarrow “(a,b,c)|d”$ ，则 $“ax+by+cz+d=0$ 有整数解” $\Leftrightarrow (a,b,c)|d$ 。$

则 $“ax+by+cz+du+e=0$ 有整数解” \Leftrightarrow “存在 u_0 使得 $(a,b,c)|(du_0+e)” \Leftrightarrow “(a,b,c)x+du+e=0$ 有整数解” $\Leftrightarrow ((a,b,c),d)|e” \Leftrightarrow (a,b,c,d)|e$ ”。即有 $“ax+by+cz+du+e=0$ 有整数解” $\Leftrightarrow (a,b,c,d)|e$ ”。同样的道理， $“a_1x_1+a_2x_2+a_3x_3+\dots+a_nx_n+b=0$ 有整数解” $\Leftrightarrow (a_1,a_2,a_3,\dots,a_n)|b$ ”。【注：每一次都用了上一次的结论，相当于在迭代。】

②.另外地，也可以纯粹地用圆向量解释这个关系： $“a_1x_1+a_2x_2+a_3x_3+\dots+a_nx_n+b=0$ 有整数解” \Leftrightarrow 存在步数组合 (x_2,x_3,\dots,x_n) 使得 $(a_2x_2+a_3x_3+\dots+a_nx_n+b)+a_1x_1=0 \Leftrightarrow$ 存在步数组合 (x_2',x_3',\dots,x_n') 使得 $((a_1,a_2)x_2'+(a_1,a_3)x_3'+\dots+(a_1,a_n)x_n'+b)+a_1x_1'=0 \Leftrightarrow$ 存在步数组合 (x_2',x_3',\dots,x_n') 使得 $((a_1,a_2)x_1'+((a_1,a_2),(a_1,a_3))x_3'+\dots+(a_1,a_2),(a_1,a_n))x_n'+b)+(a_1,a_2)x_2'=0 \Leftrightarrow$ 存在步数组合 (x_2',x_3',\dots,x_n') 使得 $((a_1,a_2),(a_1,a_3))x_3'+\dots+((a_1,a_2),(a_1,a_n))x_n'+b)+(a_1,a_2)x_2'=0 \Leftrightarrow$ 此时再把 $(a_1,a_2),(a_1,a_3)$ 作为分解对象/大圆，即有 $((a_1,a_2),(a_1,a_4)),((a_1,a_2),(a_1,a_3))x_4'+\dots+(((a_1,a_2),(a_1,a_n)),((a_1,a_2),(a_1,a_3)))x_n'+b)+((a_1,a_2),(a_1,a_3))x_3'=0$ 。。。如此下去便有之。

③.稍有不同地，还将简单地有 $“a_1x_1+a_2x_2+a_3x_3+\dots+a_nx_n+b=0$ 有整数解” \Leftrightarrow 存在步数组合 (x_2,x_3,\dots,x_n) 使得 $(a_2x_2+a_3x_3+\dots+a_nx_n+b)+a_1x_1=0 \Leftrightarrow$ 存在步数组合 (x_2',x_3',\dots,x_n') 使得 $(a_1,a_2)x_2'+(a_1,a_3)x_3'+\dots+(a_1,a_n)x_n'+b=0$ 【注： a_1x_1 总可被表示

为 $(a_1, a_2)x_2' + (a_1, a_3)x_3' + \dots + (a_1, a_n)x_n'$, 那么再合并同类项即有之 $\leftarrow \leftarrow$ 】 \Leftrightarrow 存在步数组 $(x_2', x_3', \dots, x_n')$ 使得 $((a_1, a_2), (a_1, a_3))x_3' + \dots + ((a_1, a_2), (a_1, a_n))x_n' + b = 0 \Leftrightarrow \dots$

④.另外地, 更简单地有, 因为 $(a, b)|(ax+by)$, 且因 (a, b) 为“公因数之中的最大”, 所以对于任意 z , 总存在 x, y , 可以将 $(a, b)z$ 表示为: $(a, b)z = k(a, b) = ax + by$ 【这也是之前圆向量应给出的结论之一】; 另一方面, 由于 $(a, b)|ax$, $(a, b)|by$ 所以对于任意 x, y , 总存在 z , 可以将 $ax + by$ 表示为: $ax + by = k(a, b) = (a, b)z$. 所以将其应用到 $a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + b = 0$ 中便有: $a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + b = 0 \Leftrightarrow (a_1, a_2)y_1 + a_3x_3 + \dots + a_nx_n + b = 0 \Leftrightarrow ((a_1, a_2), a_3)y_2 + a_4x_4 + \dots + a_nx_n + b = 0 \dots$

(2).正文: 如何求出其中的一个解: “超级辗转相除法”:

①.每一步的系数的最小值均唯一时的特解:

对于一个 n 元一次不定方程, 我们将其写为 $f_0(x) + b = a_1x_1 + a_2x_2 + \dots + a_nx_n + b = 0$, 其中记 $x_i = f_i(x)$. 对这 n 项里的第 i 项 a_i0 , 设第一次操作“超级辗转相除法”后, 所保留的系数 $a_{i1} = a_i0 \% \min(a_{10}, a_{20}, \dots, a_{n0})$, 现在设只有唯一的一个项的系数 $a_k(0)0 = \min(a_{10}, a_{20}, \dots, a_{n0})$, 则 $a_{i1} = a_i0 \% a_k(0)0$, 而抛开余数, 剩下的商 $= [a_i0 - a_i0 \% \min(a_{10}, a_{20}, \dots, a_{n0})] / \min(a_{10}, a_{20}, \dots, a_{n0}) = [a_i0 - a_{i1}] / \min(a_{10}, a_{20}, \dots, a_{n0}) = [a_i0 - a_{i1}] / a_k(0)0$, 那么我们有: 对于 $i \neq k(0)$, $a_{i1} = a_i0 \% a_k(0)0$, $f_{i1}(x) = f_i(x)$; $a_k(0)1 = a_k(0)0$, $f_k(0)1(x) = f_k(0)0(x) + \sum_{i=1}^n \frac{a_i0 - a_{i1}}{a_k(0)0} \cdot f_i(x)$.

若仅讨论每一步均有【 $a_k(j)j = \min(a_{1j}, a_{2j}, \dots, a_{nj})$ 】唯一, 即每一步均只有一个 $k(j)$, [此粗方括号中的 j 其实最好得写为 $j-1$ 】, 则对于用 $j-1$ 次操作后得到的下角标 2 为 $j-1$ 的方程预测下角标 2 为 j 的方程组各项系数部分和非系数部分: $i \neq k(j-1)$, $a_{ij} = a_{ij-1} \% a_k(j-1)j-1$, $f_{ij}(x) = f_{ij-1}(x)$; $a_k(j-1)j = a_k(j-1)j-1$, $f_k(j-1)j(x) = f_k(j-1)j-1(x) + \sum_{i=j}^n \frac{a_{ij-1} - a_{ij}}{a_k(j-1)j-1} \cdot f_{ij-1}(x)$. 若当 $j=m$ 时, 发现有 $a_{im} = a_{im-1} \% a_k(m-1)m-1 = 1$ 【本身是 $= (a_1, a_2, \dots, a_m)$, 这里默认之前的方程可解且已化简】【或者 $= -1$ (这是指不包括其前面的负号的 $a_{im} = 1$ or 包括前面的负号的 a_{im} 为 -1), 不过我们最好在一开始就调整方程使得不包括前面的符号的 a_{ij} 均为正的, 且连接符号也全为加号(或者说包括前面的符号的 a_{ij} 均为正的), 而负号全由 $f_{ij}(x)$ 分担, 这样对之后的 $f_{ij}(x)$ 计算也有简化作用。】, 则总共操作了 m 次后/再只进行此第 m 次操作后, 操作结束。【或者说若发现有 $a_{im+1} = a_{im} \% a_k(m)m = 0$, 则不再进行这第 $m+1$ 次操作】通过这 m 次“超级辗转相除法”操作, 我们便得到了由 n 个外带系数 a_{im} 的多项式 $f_{im}(x)$ 构成的多项式 $f_0m(x)$ 。

这个多项式 $f_0m(x)$ 有以下特点: 由 i 从 $1 \sim n$ 的 $f_{im}(x)$ 中的有关 $x_1, x_2, x_3 \dots x_n$ 的系数所构成的 n 阶行列式的值恒等于 ± 1 , 由克莱姆法则, 不论 $f_0m(x)$ 中的常数项 b ,

是如何被整数地分配进入 $f_0m(x)$ 中的各 $f_{im}(x)$ 的, 令这 n 个多项式均 $=0$ 【令它们为其他值总可以等效于刚才的分配, 所以这里不再需要解方程了】, 则该 n 个 n 元一次方程所构成的 n 阶方程组均有对应的整数解们。【注: 分配系数也相当于是解不定方程, 即 $a_{1m} \cdot f_{1m}(x) + a_{2m} \cdot f_{2m}(x) + \dots + b = 0$, 其中 a_{im} 相当于 a_i , $f_{im}(x)$ 相当于 x_i , 不过 a_i (即 a_{im}) 中有等于 1 的存在, 所以很好解出至少一组不全为 0 的解 [比如, 最好的: 令某个 (不排除有多个 $=1$ 的 a_i) $a_i=1$ 的对应的 $x_i=(-b)$ (或者说成多项式 x_i 被分配给了 b), 其余 x_i 均为 0], 即你可以对于任意 $a_i \neq 1$ 所对应的 x_i 随便赋值 (或者说随便分配给它 $(-x_i)$), 然后再用 $a_i=1$ 所对应的 x_i 赋值为 $\{(-b) - \sum \text{其它 } a_i \cdot x_i\}$ (或者说分配给它 $\{b - \sum \text{其它 } a_i \cdot (-x_i)\}$); 在此之后由于还要解一个 n 阶方程组和 n 个 n 阶方程组 [若按照之前的蓝色 [] 进行, 则化简后是: n 个 $n-1$ 阶方程组], 所以此方法一般只用于找一组特解, 找到某组特解后用其他方法寻找其他组整数解。】

②. 每一步的系数的最小值均唯一时的通解:

在有了一组特解后, 为求其通解: 对于原初的不定方程所对应的 $a_{10}x_1 + a_{20}x_2$

$+ \dots + a_{n0}x_n = 0$, 我们创造这样的 n 阶行列式:
$$\begin{vmatrix} a_{10} & a_{20} & \dots & a_{n0} \\ a_{10} & a_{20} & \dots & a_{n0} \\ c_{11} & c_{12} & \dots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-2)1} & c_{(n-2)2} & \dots & c_{(n-2)n} \end{vmatrix}$$
, 这个被

构造出来的行列式值 $=0$, 若再将其展开便有关于 a_{i0} 及其 $n-1$ 阶代数余子式的等式:

$$a_{10} \cdot \begin{vmatrix} a_{20} & a_{30} & \dots & a_{n0} \\ c_{12} & c_{13} & \dots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-2)2} & c_{(n-2)3} & \dots & c_{(n-2)n} \end{vmatrix} + a_{20} \cdot \begin{vmatrix} a_{10} & a_{30} & \dots & a_{n0} \\ c_{11} & c_{13} & \dots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-2)1} & c_{(n-2)3} & \dots & c_{(n-2)n} \end{vmatrix} + \dots + a_{n0} \cdot (-1)^{1+n} \cdot \begin{vmatrix} a_{10} & a_{20} & \dots & a_{(n-1)0} \\ c_{11} & c_{12} & \dots & c_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-2)1} & c_{(n-2)2} & \dots & c_{(n-2)(n-1)} \end{vmatrix} = 0$$

即这些整数解 (x_1, x_2, \dots, x_n) 中的某个 x_i 便 $= a_{i0}$ 的代数余子式。此时若对这个关于 c_{ij} 的 $(n-2) \times n$ 的二维数组进行全部初始化为任意值, 则每个 i 下的 a_{i0} 的代数余子式, 便全部悉数出来了, 即得到了这些 c_{ij} 们映射得到的所有的特定 i 所对应的 x_i 的整数解。于是将方程式 $a_{10}x_1 + a_{20}x_2 + \dots + a_{n0}x_n = 0$ 的各组合解 $(x_1', x_2', \dots, x_n')$ 与 $a_{10}x_1 + a_{20}x_2 + \dots + a_{n0}x_n + b = 0$ 的特定解 (x_1, x_2, \dots, x_n) 相加, 得到新的解 $(x_1, x_2, \dots, x_n) = (x_1 + x_1', x_2 + x_2', \dots, x_n + x_n')$ 。且总的 $(x_1', x_2', \dots, x_n')$ 的组合数量等于 $(x_1 + x_1', x_2 + x_2', \dots, x_n + x_n')$ 的组合数量, 即整数解组的组数。

另外地, 如果你将方程 $a_{10}x_1 + a_{20}x_2 + \dots + a_{n0}x_n = 0$ 也化为 $a_{1m} \cdot f_{1m}(x) + a_{2m} \cdot f_{2m}(x) + \dots + a_{nm} \cdot f_{nm}(x) = 0$, 那么接下来相当于 $b=0$, 你在尝试分配你的 $b=0$ 。若你再用之前求特解的方法去求一组合 $(x_1', x_2', \dots, x_n')$ 的话, 虽然之后也是涉及到跟上一个方法一样同阶的 $n-1$ 阶行列式的求解, 但是你在之前还得一次次地赋值不定方程 $a_{1m} \cdot f_{1m}(x) + a_{2m} \cdot f_{2m}(x) + \dots + a_{nm} \cdot f_{nm}(x) = 0$ 的特解 $\{f_{im}(x)\}$ 们, 既多了一步, 又属于浪费的做法: 你之前求一个特解的时候何尝不直接多求点特解, 即多种方法分解 b 或

者叫赋值？这两者是等价的。所以我们之前说过，不会再次采用之前的求特解的方法来求一般解 $(x_1', x_2', \dots, x_n')$ 们。

③. Generally, 每一步的系数的最小值可不唯一时的特解和通解：

之前的讨论建立在每一步只存在唯一的最小系数的基础之上，即 1. 刚开始的系数之中最小值只有一个，2. 之后每次(比如第 $j+1$ 次的)各个非最小值的 a_{ij} 除以这第 j 个方程(注：这里有第 0 个方程的说法)中系数的最小值时，不会出现任何“同余出下一次操作时的最小值”现象(可以出现同余)。现在我们可以去掉这项约束我们的条件了：当在任何一步的操作过程中出现“最小值不唯一”时，你可以将 $\sum_{i=j}^n \frac{a_{ij}-1-a_{ij}}{a_{k(j-1)j-1}} \cdot f_{ij} - 1(x)$ 这些礼物中的任意一个 $\frac{a_{ij}-1-a_{ij}}{a_{k(j-1)j-1}} \cdot f_{ij} - 1(x)$ ，以 $f_{ij} - 1(x)$ 为(最小)单位任意划分成“系数的最小值有多少个”所对应的多少份，然后分配给这些最小值系数们所对应的 $f_{k(j-1)j-1}(x)$ ，得到它们在第 j 个方程中的非系数部分 $f_{k(j-1)j-1}(x)$ 。接着它们的系数部分在下一个方程，即第 j 个方程中多半会转换角色，从接收方变成作为输出方将非系数分配给下一次的系数最小方们。【注意：1. 任何步骤中，所有系数非最小方均为输出方，其余均为接收方 2. 任何步骤的所有输出方必须只向系数最小方输出 3. 任何步骤的所有输出方必须以最小值的系数作为(最小)单位一份一份地输出自己的非系数部分(但体现在系数部分的削减)，且输出殆尽，即系数部分只留余数，即小于接收方的系数的数。4. 接收方们在接收时必须以 $f_{ij} - 1(x)$ 为(最小)单位在接收方内部分配。】这个法则能够保证其他更一般的多项式系数所对应的多项式，能最终得到符合期望的目标多项式 $f_{0m}(x)$ 。

根据这个规则，我们可以把每次得到的 $\frac{a_{ij}-1-a_{ij}}{a_{k(j-1)j-1}} \cdot f_{ij} - 1(x)$ 只给系数最小方中的任意一个。根据这个想法，再加上之前求特解时的[最好的：令某个(不排除有多个=1的 a_i) $a_i=1$ 的对应的 $x_i=(-b)$ ，其余 x_i 均为 0]易行规则，我用它们编制了这样的规则下的 c 语言代码。在代码的调试过程中，我发现我的理论有个非常大的漏洞：当某一步出现了有输出方能够整除接收方时，整个程序和我的理论便无法继续进行操作了：我的理论规定了所有的系数非最小方均为输出方，这是致命的，因为接下来要么继续操作，则有些项的系数部分会成 0，要么不继续操作，则程序终止。

不过我突发奇想，只需要稍稍修正一下“对输出方的定义”，就可以使整个理论自治地能够处理任何所输入的数字在运行过程中的任何可能的步骤了，并且让理论非常漂亮：所有的系数不能整除系数最小方的系数的项，均为输出方(相对于以前，能成为输出方的条件更严格了)，接收方的定义仍然不改变：所有的系数最小方，均为接受方。这样一来由于它们并集不为全，所以有一些既不为输出方也不为接收方的存在：即所有的系数非最小方当中，那些系数能整除系数最小方的系数的项。有意思的是如果随着进程的继续深入，导致除了系数最小方以外，其它项均为此类项的话，那么程

序工作就已经完成(没有输出方了! 系数最小方一直在等待被赠予非系数部分), 即系数最小方的系数, 已经成为各个阶段的各项的系数部分的最大公因数了; 另外地, 新理论还可以用“输出方”解释“接受方”: 系数最小方可能不止一个, 由于系数最小方的系数=系数最小方的系数, 所以系数最小方的系数能整除系数最小方的系数, 所以系数最小方不是输出方, 而又因为它们是“不是输出方中的最小的”, 所以它们是非输出方中的: 接受方。这便是之前没能解释为什么到此为止、以及规则成因的, 真正解释, which 使我们从非真理直接晋升为了真理。

3.程序: 找一组 n 元一次不定方程的整数解

那么, 这个用于找一组 n 元一次不定方程的整数解的代码如下: (vc++6.0, 250 行, 10 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

int getmin[2],value1=1,sign=1,value2=0;          /*创建一个有两元素的一维
全局数组变量 getmin[2],一个累乘器,一个对换装置,一个累加器.*/

void minandAddress(int *p,int count)              /*创建一个无返回值函数,获取
一个地址和该地址下的 int 类地址个数*/
{
    int i;
    getmin[0]=p[0];
    getmin[1]=0;
    for(i=1;i<count;i++)
    {
        if(p[i]<getmin[0])                        /*这里确定了最小值将是第
一个最小值,因为如果之后的 p[i]=getmin[0],则 getmin[1]不会改变*/
        {
            getmin[0]=p[i];
            getmin[1]=i;
```

```

    }
}
}
int maxjoint(int *p,int count)                /*创建函数 maxjoint 调取一串整数,
返回它们的最大公因数*/
{
    int j,i,t,*pp=(int*)malloc(count*sizeof(int));    /*p2 这个地址是共用的,为
    了保持 p2 内容(即实验者之前的输入顺序)不变,创建一个 p2 的复制品*/

    for(i=0;i<count;i++)                        /*初始化 p2 的复制品*/
        pp[i]=p[i];

    for(j=count-1;j>0;j--)                        /*刚开始就调换好所有相邻数据
    的大小顺序以便之后的辗转相除*/
    {
        for(i=0;i<j;i++)
        {
            if(pp[i]<pp[i+1])
            {
                t=pp[i];
                pp[i]=pp[i+1];
                pp[i+1]=t;
            }
        }
    }

    for(i=count-2;i>=0;i--)                        /*由于之前是从左往右==从大
    到小,那么此时得从后往前轮换*/
    {
        do
        {
            t=pp[i]%pp[i+1];

```

```

        pp[i]=pp[i+1];
        pp[i+1]=t;
    }

    while (t!=0);                                /*跳出循环后 pp[i]就是 pp[i]
和 pp[i+1]的最大公因,而剩下的所有右边的 pp[i+1]会全是 0*/

    }
    t=pp[0];
    free(pp);                                    /*不用的早点释放掉*/
    pp=NULL;
    return t;
}

void initialisep4(int *p4,int count)              /*本来以为 matrix 函数会改变 p4
的排序状况,因而想设置它来每次调用它来初始化 p4,多心了*/
{
    int i;
    for(i=0;i<count;i++)
        p4[i]=i;
}

void exchange(int *a,int *b)
{
    int t=*a;
    *a=*b;
    *b=t;
}

void matrix(int *j,int min,int max,int **p3)      /*输入一组关于列的下角标 j 排
列的地址,地址起始偏移量,地址最末偏移量,用于计算的行列式首地址*/
{
    if(min==max)                                /*丰收的标志:该收割此次收获了
*/

```

```

{
    for(int k=0;k<=max;k++)
        value1*=p3[k][j[k]];           /*累乘器起作用了*/
    value2+=value1*sign;                /*累加器起作用了*/
    value1=1;                          /*初始化累乘器*/
}
else
{
    for(int i=min;i<=max;i++)
    {
        exchange(j+min,j+i);
        if(i!=min)                    /*不和自己对换的对换称
为有效对换,当有效对换一次时,排列改变奇偶性*/
            sign=-1*sign;
        matrix(j,min+1,max,p3);
        exchange(j+min,j+i);
        if(i!=min)                    /*好东西在这里也得写,不
然就成为坏东西了*/
            sign=-1*sign;
    }
}
}
int matrixoutput(int *j,int min,int max,int **p3,int **pp3,int *b,int i,int k)
{
    int jj,ii;
    value2=0;                          /*初始化 value2*/
    for(jj=0;jj<=max;jj++)             /*初始化 pp3 成 p3 的样子*/
        for(ii=0;ii<=max;ii++)

```

```

        pp3[ii][jj]=p3[ii][jj];
/*      for(ii=0;ii<=max;ii++)                打印原始系数矩阵
{
    for(jj=0;jj<=max;jj++)
        printf("%4d",pp3[ii][jj]);
    printf("\n");
}*/

    for(ii=0;ii<=max;ii++)                    /*用 b 代换后的系数矩阵*/
        pp3[ii][k]=0;
    pp3[i][k]=b[i];
/*      printf("\n");                        打印它
    for(ii=0;ii<=max;ii++)
    {
        for(jj=0;jj<=max;jj++)
            printf("%4d",pp3[ii][jj]);
        printf("\n");
    }*/
    matrix(j,min,max,pp3);
    return value2;
}

/*注:p2 有 count2+1 个空间;有 count2 个数;有 count2-1 个 ai 系数;count=count2-1
即系数个数;count-1 即末位系数的数组序数(下标)*/

int main()
{
    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*pp2,s,count1=0,count2=0,i,sum,j,getmax,*b,*p
    4,value2copy,**pp3,flag=1;

```

```

printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

s=getchar();                                /*创建一个 scanf 函数用于实现我
的功能,刚开始就需要值进来,所以不用 dowhile*/

while(s!='\n')
{
    if(s<=57&&s>=48)
    {
        count1++;                            /*统计输入了多少个"
假·数字"进来*/

        *(p1+count1-1)=s;                    /*将 s 的值放入距离
p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

        p1=(char*)realloc(p1,count1+1);      /*将 p1 的空间扩
张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/
    }
    else if(s=='-')
        flag=-1;                            /*只要输入了负号,不管输
入了多少个,逗号与逗号之间的数均判定为负.*/

    else if(s==',')
    {
        sum=0;
        /*****初始化 sum 值,为了
下面的循环*/

        for(i=0;i<=count1-1;i++)
            sum=sum+(*(p1+i)-'0')*(int)pow(10,count1-1-i);

        if(flag==-1)
        {
            sum=-sum;
            flag=1;

```



```

    }
    count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

    count2++;                      /*统计输入了多少个"
真·数字"进来*/

    *(p2+count2-1)=sum;            /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

    p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

    }
    s=getchar();

}
if(count2-1<2)
{

    printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

    return 0;

}

p3=(int**)malloc((count2-1)*sizeof(int*)); /*除 b 以外有(count2-1)
个系数,所以只需给二维数组的首地址 p3 分配那么多空间,因为只有那么多个非系数部
分(行/一维数组)*/

for(i=0;i<count2-1;i++)
{

    p3[i]=(int*)malloc((count2-1)*sizeof(int)); /*除 b 以外有
(count2-1)个系数,所以只需给一维数组的首地址 p3[i]分配那么多空间,因为只有那么多
个非系数部分里的非系数部分(列/一维数组里的元素)*/

    for(j=0;j<count2-1;j++)

```

```

        if(j==i)                                /*趁此机会把非系数部分的
值给赋了,和下面构成系数部分和非系数部分之内外负号调换*/

        if(p2[i]>0)
            p3[i][j]=1;
        else p3[i][j]=-1;
        else p3[i][j]=0;

        p2[i]=abs(p2[i]);                        /*趁此机会把系数部分取绝
对值,和上面构成系数部分和非系数部分之内外负号调换*/
    }

    getmax=maxjoint(p2,count2-1);                /*在用 maxjoint 之前得保
证系数是正的,所以 p3 的创造和赋值也得在之前*/

    if(p2[count2-1]%getmax!=0)
    {

        printf("您输入的多元一次不定方程无整数解。 \n");
        for(i=0;i<count2-1;i++)
        {
            free(p3[i]);
            p3[i]=NULL;
        }
        free(p3);
        p3=NULL;
        free(p2);
        p2=NULL;
        free(p1);
        p1=NULL;
        return 0;
    }

    sum=1;                                        /*创建跳出循环的条件:只要这一群
系数中有一个等于上一群(=最开始的)系数的最大公因数,则跳出*/

```

```

pp2=(int*)malloc((count2-1)*sizeof(int));          /*复制一个 p2 的镜像,短
暂保存当前的 p2 下的各地址的值*/

minandAddress(p2,count2-1);                          /*传计算出来后保存至所开
辟的两个空间里的值给 getmin*/

do                                                    /******进入数学部分
*****/
{
    for(i=0;i<count2-1;i++)                          /*好吧,为了之后的简洁,
我们还是全部初始化好了*/
        pp2[i]=p2[i];

    for(i=0;i<count2-1;i++)                          /*先搞定系数部分,此过程
中不能动最小系数方的非系数部分*/
    {
        if(p2[i]%getmin[0]!=0)                      /******→→→
→就是这里,得把原来的 p2[i]!=getmin[0]改为 p2[i]%getmin[0]!=0,即从"不等于"改为
"不整除"为输出方*****/

/*                pp2[i]=p2[i];                初始化 p2 的复制品,
将此语句放入此 if 下,是因为最小系数方的非系数部分只接受给予方的非系数部分,所以
没必要将 p2 下的内容全拿给 pp2*/

        p2[i]=p2[i]%getmin[0];                      /*若为非最小值,
即为输出方,即系数部分输出,而非系数部分不改变;最小值方们系数部分均未改变*/

    }

    for(j=0;j<count2-1;j++)                          /*搞定非系数部分*/
    {
        for(i=0;i<count2-1;i++)
        {
            p3[getmin[1]][j]=p3[getmin[1]][j]+(pp2[i]-
p2[i])/getmin[0]*p3[i][j];
        }
    }
}

```

```

    }                                     /*当
p2[i]==getmin[0]&&!=getmin[1]时,系数部分和非系数部分均不发生改变.此时并集已
经为全.*****→→→→修正为:满足 p2[i]%getmin[0]==0&&!=getmin[1]的这
些项的两部分均不改变*****/

    minandAddress(p2,count2-1);

    if(getmin[0]==getmax)                 /******→→→→
根据修正后的观念,这句话既可以按原来的翻译,也可以翻译为:如果所有项的系数部分
均能整除它们的最大公因数,那么根据理论这些项的两部分均不改变,则程序的 mission
已经 accomplished*****/

    {
        sum=0;
        b=(int*)malloc((count2-1)*sizeof(int));
        for(i=0;i<count2-1;i++)
            b[i]=0;

        b[getmin[1]]=p2[count2-1]/getmax;    /*给这个最后的
最小系数方的非系数部分的 b 赋值为原不定方程中的 b*/

    }

    }

    while(sum);

    p4=(int*)malloc((count2-1)*sizeof(int));
    initialisep4(p4,count2-1);
    matrix(p4,0,count2-2,p3);
    value2copy=value2;

    pp3=(int**)malloc((count2-1)*sizeof(int*));    /*创建一个 p3 的 copy,
用于将常数项 b[i]替换掉系数矩阵中的第 j 列矢量*/

    for(i=0;i<count2-1;i++)
        pp3[i]=(int*)malloc((count2-1)*sizeof(int));

    for(j=0;j<count2-1;j++)                    /*输出*/

    {

```

```
printf("x%d=%d\n",j+1,matrixoutput(p4,0,count2-2,p3,pp3,b,getmin[1],j)/value2copy);
```

/* for(i=0;i<count2-1;i++) 用于调试中检查程序运行情况.

```
printf("%d",p4[i]);
printf("\n%d\n",sign);*/
}
for(i=0;i<count2-1;i++) /*弹出*/
{
    free(pp3[i]);
    pp3[i]=NULL;
}
free(pp3);
pp3=NULL;
free(p4);
p4=NULL;
free(b);
b=NULL;
free(pp2);
pp2=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;
}
free(p3);
p3=NULL;
free(p2);
p2=NULL;
```

```

free(p1);
p1=NULL;
}

```

以上代码在程序核心：解方程组的时候，未采用主元素的高斯消去法(避免浮点数运算的不尽人意)，而采用的非“算逆序数”的，而是“**标记对换次数情况**”的方法，与**与其相适应的递归算法之寻找全排列**的进程**同时进行**，运算量上大大减小【但运算量的绝对值仍然非常巨大，相对于接下来的方法。所以不推荐用这种方法计算系数(元数)超过 10 个的多元一次不定方程的特解。(至少我的电脑在计算加上 b 共 12 个数的不定方程的时候，计算第一个解花了大约 10s，然后剩下的解大约 7s 出一个...)】，而且非常贴近人工/数学思维。

这个长达 10 页的代码就算是我的理论和规则的具体细节化，一切已提及的和尚未提及的方面，都广义地涵盖在里面了，如若能明白代码所想传达的意义，那么这套理论也就不过如此了。

接下来，我将用另一个理论，附带另一则与之对应的代码，引入另一套，更通俗易懂地且更能简便地，求得不定方程解的通式的理论(后者主攻方向为通式，前者主攻方向为个性解；**虽然前者也能求通解**，只需要将“b[getmin[1]]=p2[count2-1]/getmax;”所在的复合语句块改一改，或者更好地，另设一程序，只利用以上代码中的求解行列式部分，引用之前提到的这种特解所对应的“通解理论”，但它无法保证不同随机的种子的排列组合下的解一定不同；**虽然后者也能求特解**，但后者都既然能够用程序输出用字母表示的通解了，何必只用它输出一个个特解呢)。

4.程序：给定系数的各阶多元一次不定方程的通解

i.理论部分：

这个方法是充分利用①.**规则**：对于任意 z ，总存在 x, y ，使得 $(a,b)z=ax+by$ 【主要是它，但本质上用的是充要条件，只不过另一个由于较简单而不再赘述】；以及唯一存在的，②.**最底层的通式**：二阶时候的 $(x_0+t*b/(a,b), y_0-t*a/(a,b))$ 或 $(x_0-t*b/(a,b), y_0+t*a/(a,b))$ 为 $ax+by+c=0$ 的通解。

a.首先先引入一个用此方法求特解的，易于理解的过程：

Part(1).对于 $a_1x_1+a_2x_2+\dots+a_nx_n=b$ ，设 $a_1x_1+a_2x_2=(a_1,a_2)y_1$ ，根据“①.**规则**”，代换后得到的 $(a_1,a_2)y_1+a_3x_3+\dots+a_nx_n=b$ ，仍可以回到 $a_1x_1+a_2x_2+\dots+a_nx_n=b$ ，因而此做法因是双向的而有效 or 可行；重复以上步骤，有 $((a_1,a_2),a_3)$

$y_2 + a_4 x_4 \dots + a_n x_n = b$, 以至 $(a_1, a_2, \dots, a_{n-1}) y_{n-2} + a_n x_n = b$. Part(2). 对 $(a_1, a_2, \dots, a_{n-1}) y_{n-2} + a_n x_n = b$ 运用吾辈开发的公式(最后却仅仅用来解 2 元不定方程还真有点大材小用啊), 返回 (x_n, y_{n-2}) 的一个特解 (x_0, y_0) , 带入②. 最底层的通式 $(x_0 + t*b/(a, b), y_0 - t*a/(a, b))$ 得 $(x_n, y_{n-2}) = (x_0 + t*(a_1, a_2, \dots, a_{n-1})/(a_1, a_2, \dots, a_n), y_0 - t*a_n/(a_1, a_2, \dots, a_n))$, 此时, 注意保留其中的 $x_n = x_0 + t*(a_1, a_2, \dots, a_{n-1})/(a_1, a_2, \dots, a_n)$, 另将其中的 $y_{n-2} = y_0 - t*a_n/(a_1, a_2, \dots, a_n)$ 代入之前上一层的 $(a_1, a_2, \dots, a_{n-2}) y_{n-3} + a_{n-1} x_{n-1} = (a_1, a_2, \dots, a_{n-1}) y_{n-2}$ 中, 赋值给 t 一个特值【如各位所见, 总将 t 赋值为 0 最易, 即对应的 $y_{n-2} = y_0 =$ 上一次解出来的 y_{n-2} 的特解】, 得到 y_{n-2} 的一个特值, 返回一个 (x_{n-1}, y_{n-3}) 的一个特解 (x_0', y_0') , 带入②. 最底层的通式得到 $(x_{n-1}, y_{n-3}) = (x_0' + t'*(a_1, a_2, \dots, a_{n-2})/(a_1, a_2, \dots, a_{n-1}), y_0' - t'*a_{n-1}/(a_1, a_2, \dots, a_{n-1}))$, 保留其中的 x_{n-1} , 将其中的 y_{n-3} 的特解再代入下一层.....一直循环这些步骤, 直至利用 y_1 的特解, 保留 x_1 和 x_2 的通式。(以上, 从红字“返回”到红字“一层”, 是一个最小正周期。)

b. 现依以上过程, 给出用此方法求通解的过程:

只需要将以上过程中的, 对 t, t' 的赋值步骤省略, 即可得到通解。我们之所以之前在 a. 部分没这么说, 是因为 t 是个变量, 所导致的 $(a_1, a_2, \dots, a_{n-2}) y_{n-3} + a_{n-1} x_{n-1} = (a_1, a_2, \dots, a_{n-1}) y_{n-2}$, 中的 y_{n-2} 是个关于 t 的函数, 而如果接下来说“返回一个 (x_{n-1}, y_{n-3}) 的一个特解 (x_0', y_0') ”, 会让人感到毫无头绪我在说什么没逻辑的话(没有特值 t 何来特解 (x_0', y_0')): 其实事实是, 每一个 t 会导致一个因 y_{n-2} 变化而产生的特解 (x_0', y_0') , 而之后将出现的 $(x_{n-1}, y_{n-3}) = (x_0' + t'*(a_1, a_2, \dots, a_{n-2})/(a_1, a_2, \dots, a_{n-1}), y_0' - t'*a_{n-1}/(a_1, a_2, \dots, a_{n-1}))$, 其中的 (x_0', y_0') 不再是特定 t 值下的特定 y_{n-2} 所算出的常量了, 而是一个关于 t 的函数。因此, (x_{n-1}, y_{n-3}) 中的两个, 都是关于 t 和 t' 的函数。因此, 只与一个变量 t 有关的特解 (x_0', y_0') , 确实是相对于与两个变量 t 和 t' 有关的 (x_{n-1}, y_{n-3}) 通式的特解。

可能你此时就会关心: 当每一层的 b 值不确定的时候, 怎么确定每一层的 $a_1 x_1 + a_2 x_2 = b$ 的具体特解呢? 没关系, 我们并不关心 b 是否有确定值, 只关心 b 是否有确定的表达式, 即 b 是个表达式(b 是个因变量)也可以, 但是得让这个表达式除因变量之外的系数是确定的。为什么呢? 这里的技术保障源于: 在我的算法里, 我们的不定方程的解(的分母)与 b 的关系不大。

ii. 代码部分: (vc++6.0, 468 行, 16 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
```

```
#include<math.h>
```

```
#include<stdlib.h>
```

其它作品下载链接. (click here) Turn up the sound. (my tunes) chzh_xie@foxmail.com. (contact)

```
int getmin[2];                                /*创建一个有两元素的一维全局数组变
量 getmin[2]*/
```

```
void minandAddress(int p2,int p22,int addressp2)
```

```
{
    getmin[0]=p2;
    getmin[1]=addressp2;
    if(p22<getmin[0])
    {
        getmin[0]=p22;
        getmin[1]=addressp2-2;
    }
}
```

```
int maxjoint(int p2,int p22)                  /*创建函数 maxjoint 调取一对值,返
回它们的最大公因数数值*/
```

```
{
    int t,p[2];
    p[0]=p2;
    p[1]=p22;
    if(p[0]<p[1])                               /*调换数据的大小顺序以便之后的
辗转相除*/
```

```
{
    t=p[0];
    p[0]=p[1];
    p[1]=t;
```

```
}
```

```
do
```

```
{
    t=p[0]%p[1];
    p[0]=p[1];
```

```

        p[1]=t;
    }

    while (t!=0);                                /*跳出循环后 p[0]就是*p2 和*pp2
    的最大公因*/

    t=p[0];
    return t;
}

/*****/

int main()
{
    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*p22,**p33,pp2[2],s,count1=0,count2=0,i,sum,k
    ,value2copy,**x,**y,*p222,*pp22,flag=1;

    printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
    a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

    s=getchar();                                /*创建一个 scanf 函数用于实现我
    的功能,刚开始就需要值进来,所以不用 dowhile*/

    while(s!='\n')
    {
        if(s<=57&&s>=48)
        {
            count1++;                            /*统计输入了多少个"
            假·数字"进来*/

            *(p1+count1-1)=s;                    /*将 s 的值放入距离
            p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

            p1=(char*)realloc(p1,count1+1);      /*将 p1 的空间扩
            张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/

        }

        else if(s=='-')

```

```

        flag=-1;
    else if(s=='')
    {
        sum=0;
/*****初始化 sum 值,为了
下面的循环*/

        for(i=0;i<=count1-1;i++)
            sum=sum+*(p1+i)-'0')*(int)pow(10,count1-1-i);
        if(flag== -1)
        {
            sum=-sum;
            flag=1;
        }
        count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

        count2++;                                /*统计输入了多少个"
真·数字"进来*/

        *(p2+count2-1)=sum;                        /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

        p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

    }
    s=getchar();
}
if(count2-1<2)
{
    printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

```

```

        return 0;
    }

    p3=(int**)malloc((count2-1)*sizeof(int));          /*从 a0 到 an-1 的非系数
部分:x0 到 xn-1*/

    for(i=0;i<count2-1;i++)

        p3[i]=(int*)malloc(2*sizeof(int));            /*现在每个子地址只开
辟两个空间了,每对新创造的最小公倍数和右系数只构成一个二元一次不定式,两个系数,
两个非系数部分.这里只给 x 所在的非系数部分分配空间.*/

        pp22=(int*)malloc((count2-1)*sizeof(int));    /*创造 pp22 来复制 p2,
保留它们的原始值,以后有用。并且要在它们的负号消失之前及时得到最原始的信息.不
能丢失此负号信息.*/

        for(i=0;i<=count2-1-1;i++)

            pp22[i]=p2[i];

/*****/

        for(i=0;i<=1;i++)                            /*先初始化 a0 和 a1 的系数部分
和非系数部分,这里可以沿用之前的语法,之后就不行了*/

        {

            for(k=0;k<=1;k++)

                if(k==i)

                    if(p2[i]>0)

                        p3[i][k]=1;

                    else p3[i][k]=-1;

                else p3[i][k]=0;

            p2[i]=abs(p2[i]);

        }

/*****/

        if(count2-1>2)

        {

```

p22=(int*)malloc((count2-1-2)*sizeof(int)); /*从 b0 到 bn-3 的系数部分,这是模仿 p2 创造的,但是 p2 还包含有 b 以及一个空空间.*/

p33=(int**)malloc((count2-1-2)*sizeof(int*)); /*从 b0 到 bn-3 的非系数部分:y0 到 yn-3*/

for(i=0;i<count2-1-2;i++)

p33[i]=(int*)malloc(2*sizeof(int)); /*其子地址也只开辟两个空间*/

}

else /*城乡结合部:综合最初(k=n-3)与最末(k=-1)二阶式子的语句,具体而微的程序雏形.*/

{

if(p2[count2-1]%maxjoint(p2[0],p2[1])!=0)

{

printf("您输入的多元一次不定方程无整数解。 \n");

free(pp22);

pp22=NULL;

for(i=0;i<count2-1;i++)

{

free(p3[i]);

p3[i]=NULL;

}

free(p3);

p3=NULL;

free(p2);

p2=NULL;

free(p1);

p1=NULL;

return 0;

}


```

x=(int**)malloc((count2-1)*sizeof(int*));
sum=1;
minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0];                                /*保留值*/
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {
        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));

```

if(pp22[0]*pp22[1]>0) /*这里才是真迹:如果系数
异号,则规定它们的最大公因数为负值.*/

```

    flag=maxjoint(p2[1],p2[0]);
else
    flag=-maxjoint(p2[1],p2[0]);
x[1][count2-2]=pp22[0]/flag;
x[0][count2-2]=-pp22[1]/flag;
if(getmin[1]==-1)
{
    for(i=count2-3;i>=0;i--)
    {
        x[1][i]=-p2[count2-1]/getmin[0]*p3[1][0]/value2copy;
        x[0][i]=p2[count2-1]/getmin[0]*p3[1][1]/value2copy;
    }
}
else
{
    for(i=count2-3;i>=0;i--)
    {
        x[1][i]=p3[0][0]*p2[count2-1]/getmin[0]/value2copy;
        x[0][i]=-p3[0][1]*p2[count2-1]/getmin[0]/value2copy;
    }
}
for(k=count2-2;k>=1;k--)
{
    printf("x%d=",k+1);
    for(i=count2-1-k;i>=1;i--)
        printf("%dt%d+",x[k][i],i);
    printf("%d",x[k][0]);
}

```

```
        printf("\n");
    }
    printf("x%d=",1);
    for(i=count2-2;i>=1;i--)
        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
    free(x[0]);
    x[0]=NULL;
    free(x[1]);
    x[1]=NULL;
    free(x);
    x=NULL;
    free(pp22);
    pp22=NULL;
    for(i=0;i<count2-1;i++)
    {
        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}
```

/*****/

p22[0]=maxjoint(p2[0],p2[1]); /*没办法,任何设计都有局限,这里多出了一些周期之外的东西,加上上面的,有两部分周期之外的,这里除了这第一句话,相当于下面循环中 k=0 的时候的.*/

```

p33[0][0]=1;
p33[0][1]=0;
p3[2][0]=0;
if(p2[2]>0)
    p3[2][1]=1;
else p3[2][1]=-1;
p2[2]=abs(p2[2]);

/*****/
for(k=1;k<=count2-1-3;k++)
{
    p22[k]=maxjoint(p2[k+1],p22[k-1]);
    p33[k][0]=1; /*p22[k]由于是上一对系数的
最大公因数,恒>0,所以直接 p33[k][0]=1,剩下的=0,以及不用取系数的绝对值了*/
    p33[k][1]=0;
    p3[k+2][0]=0;
    if(p2[k+2]>0)
        p3[k+2][1]=1;
    else p3[k+2][1]=-1;
    p2[k+2]=abs(p2[k+2]);
}

/*****/
if(p2[count2-1]%maxjoint(p2[count2-2],p22[count2-4])!=0)
{
    printf("您输入的多元一次不定方程无整数解。 \n");
    for(i=0;i<count2-1-2;i++)
    {

```

```

        free(p33[i]);
        p33[i]=NULL;
    }
    free(p33);
    p33=NULL;
    free(p22);
    p22=NULL;
    free(pp22);
    pp22=NULL;
    for(i=0;i<count2-1;i++)
    {
        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}

x=(int**)malloc((count2-1)*sizeof(int*));
y=(int**)malloc((count2-1-2)*sizeof(int*));

p222=(int*)malloc((count2-1-2)*sizeof(int));    /*创造 p222 来复制 p22,
保留它们的原始值,以后有用,注:由于 p22 全是正的,所以 p222 也全是正的.*/

for(i=0;i<=count2-1-3;i++)
    p222[i]=p22[i];

```

```

/*****/
3*/

sum=1;
minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
do
{
    pp2[0]=p22[count2-1-3];          /*保留值*/
    pp2[1]=p2[count2-1-3+2];
    if(p22[count2-1-3]%getmin[0]!=0)
        p22[count2-1-3]=p22[count2-1-3]%getmin[0];
    if(p2[count2-1-3+2]%getmin[0]!=0)
        p2[count2-1-3+2]=p2[count2-1-3+2]%getmin[0];
    if(getmin[1]==count2-1-3)
    {
        p33[count2-1-3][0]=p33[count2-1-3][0]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][0];
        p33[count2-1-3][1]=p33[count2-1-3][1]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][1];
    }
    else
    {
        p3[count2-1-3+2][0]=p3[count2-1-3+2][0]+(pp2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][0];
        p3[count2-1-3+2][1]=p3[count2-1-3+2][1]+(pp2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][1];
    }
    minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
    if(getmin[0]==maxjoint(p2[count2-1-3+2],p22[count2-1-3]))
        sum=0;
}
while(sum);

```

```

    value2copy=p33[count2-1-3][0]*p3[count2-1-3+2][1]-p33[count2-1-3][1]*p3[count2-1-3+2][0];
    x[count2-1-3+2]=(int*)malloc(2*sizeof(int));
    y[count2-1-3]=(int*)malloc(2*sizeof(int));

    if(pp22[count2-1-3+2]>0)                                /*这里本来是判断
    p222[count2-1-3]*pp22[count2-1-3+2]>0,但由于 p222[count2-1-3]本身就已经人为
    规定>0 了,所以能让程序跑快点就让它跑快点嘛.*/
        flag=maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    else
        flag=-maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    x[count2-1-3+2][1]=p222[count2-1-3]/flag;
    y[count2-1-3][1]=-pp22[count2-1-3+2]/flag;
    if(getmin[1]==count2-1-3)
    {
        x[count2-1-3+2][0]=-p2[count2-1]/getmin[0]*p3[count2-1-3+2][0]/value2copy;
        y[count2-1-3][0]=p2[count2-1]/getmin[0]*p3[count2-1-3+2][1]/value2copy;
    }
    else
    {
        x[count2-1-3+2][0]=p33[count2-1-3][0]*p2[count2-1]/getmin[0]/value2copy;
        y[count2-1-3][0]=-p33[count2-1-3][1]*p2[count2-1]/getmin[0]/value2copy;
    }
    /*****/

    for(k=count2-1-3-1;k>=0;k--)
    {
        sum=1;
        minandAddress(p2[k+2],p22[k],k+2);
    }

```



```

do
{
    pp2[0]=p22[k];                /*保留值*/
    pp2[1]=p2[k+2];
    if(p22[k]%getmin[0]!=0)
        p22[k]=p22[k]%getmin[0];
    if(p2[k+2]%getmin[0]!=0)
        p2[k+2]=p2[k+2]%getmin[0];
    if(getmin[1]==k)
    {
        p33[k][0]=p33[k][0]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][0];
        p33[k][1]=p33[k][1]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][1];
    }
    else
    {
        p3[k+2][0]=p3[k+2][0]+(pp2[0]-
p22[k])/getmin[0]*p33[k][0];
        p3[k+2][1]=p3[k+2][1]+(pp2[0]-
p22[k])/getmin[0]*p33[k][1];
    }
    minandAddress(p2[k+2],p22[k],k+2);
    if(getmin[0]==maxjoint(p2[k+2],p22[k]))
        sum=0;
}
while(sum);
value2copy=p33[k][0]*p3[k+2][1]-p33[k][1]*p3[k+2][0];
x[k+2]=(int*)malloc((count2-1-k-1)*sizeof(int));
y[k]=(int*)malloc((count2-1-k-1)*sizeof(int));
if(pp22[k+2]>0)

```

```

        flag=maxjoint(p2[k+2],p22[k]);
    else
        flag=-maxjoint(p2[k+2],p22[k]);
    x[k+2][count2-1-k-2]=p222[k]/flag;
    y[k][count2-1-k-2]=-pp22[k+2]/flag;
    if(getmin[1]==k)
    {
        for(i=count2-1-k-3;i>=0;i--)
        {
            x[k+2][i]=-
y[k+1][i]*p222[k+1]/getmin[0]*p3[k+2][0]/value2copy;

            y[k][i]=y[k+1][i]*p222[k+1]/getmin[0]*p3[k+2][1]/value2copy;
        }
    }
    else
    {
        for(i=count2-1-k-3;i>=0;i--)
        {

            x[k+2][i]=p33[k][0]*y[k+1][i]*p222[k+1]/getmin[0]/value2copy;
            y[k][i]=-
p33[k][1]*y[k+1][i]*p222[k+1]/getmin[0]/value2copy;
        }
    }
}

/*****/          /*相当于 k=-1*/

sum=1;
minandAddress(p2[1],p2[0],1);
do
{

```

```

pp2[0]=p2[0];                                /*保留值*/
pp2[1]=p2[1];
if(p2[0]%getmin[0]!=0)
    p2[0]=p2[0]%getmin[0];
if(p2[1]%getmin[0]!=0)
    p2[1]=p2[1]%getmin[0];
if(getmin[1]==1-2)
{
    p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
    p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
}
else
{
    p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
    p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
}
minandAddress(p2[1],p2[0],1);
if(getmin[0]==maxjoint(p2[1],p2[0]))
    sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));

if(pp22[0]*pp22[1]>0)                        /*这里才是真迹:如果系数异号,
则规定它们的最大公因数为负值.*/
    flag=maxjoint(p2[1],p2[0]);
else
    flag=-maxjoint(p2[1],p2[0]);

```

```

x[1][count2-2]=pp22[0]/flag;
x[0][count2-2]=-pp22[1]/flag;
if(getmin[1]==-1)
{
    for(i=count2-3;i>=0;i--)
    {
        x[1][i]=-y[0][i]*p222[0]/getmin[0]*p3[1][0]/value2copy;
        x[0][i]=y[0][i]*p222[0]/getmin[0]*p3[1][1]/value2copy;
    }
}
else
{
    for(i=count2-3;i>=0;i--)
    {
        x[1][i]=p3[0][0]*y[0][i]*p222[0]/getmin[0]/value2copy;
        x[0][i]=-p3[0][1]*y[0][i]*p222[0]/getmin[0]/value2copy;
    }
}

/*****/

for(k=count2-2;k>=1;k--)
{
    printf("x%d=",k+1);
    for(i=count2-1-k;i>=1;i--)
        printf("%dt%d+",x[k][i],i);
    printf("%d",x[k][0]);
    printf("\n");
}
printf("x%d=",1);
for(i=count2-2;i>=1;i--)

```

```

        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
/*****/

    free(x[0]);
    x[0]=NULL;
    free(x[1]);
    x[1]=NULL;
    for(k=0;k<=count2-1-3-1;k++)
    {
        free(y[k]);
        y[k]=NULL;
        free(x[k+2]);
        x[k+2]=NULL;
    }
    free(y[count2-1-3]);
    y[count2-1-3]=NULL;
    free(x[count2-1-3+2]);
    x[count2-1-3+2]=NULL;
    free(p222);
    p222=NULL;
    free(y);
    y=NULL;
    free(x);
    x=NULL;
    for(i=0;i<count2-1-2;i++)
    {
        free(p33[i]);
        p33[i]=NULL;
    }

```

```
}  
free(p33);  
p33=NULL;  
free(p22);  
p22=NULL;  
free(pp22);  
pp22=NULL;  
for(i=0;i<count2-1;i++)  
{  
    free(p3[i]);  
    p3[i]=NULL;  
}  
free(p3);  
p3=NULL;  
free(p2);  
p2=NULL;  
free(p1);  
p1=NULL;  
}
```

二.多项式定理的计组合数问题

在上一本书中，在“多项式定理”一节的末尾，和“复合函数的高阶导”一节的末尾，均出现了要穷尽某些条件下的，对应的某些不定方程的解或者解的组合数，这一类问题。由于在写这本书时，我恰好学了c语言，所以我将这些理论分别写成程序放在这本书中：

1.程序：多项式定理的组合数问题的所有解

期待已久的，理论的代码形式：(vc++6.0, 94 行, 3 页 word 文档, 未注明各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

int
j=0,count=0,*hang=(int*)malloc(sizeof(int)),mohang,*lie=(int*)malloc(sizeof(int));

int min(int m,int n)
{
    int t;
    t=m;
    if(n<m)
        t=n;
    return t;
}

void fun(int m,int n)
{
    int i,m1,n1,k;
    if(n>1)
    {
        for(i=0;i<=n-1;i++)
        {
            j++;
            n1=n-i;          /*valid*/
            hang=(int*)realloc(hang,j*sizeof(int));
            hang[j-1]=n1;
            m1=m-n1;         /*valid*/
        }
    }
}
```



```

        mohang=m1;
        n1=min(m1,n1);    /*valid*/
        fun(m1,n1);
        j--;
    }
}
else
{
    count=count+1;
    if(n==1)
    {
        hang=(int*)realloc(hang,(j+mohang)*sizeof(int));
        for(i=j;i<=j+mohang-1;i++)
            hang[i]=1;
        /**/
        printf("按行从上到下计数:");
        for(i=j+mohang-1;i>=0;i--)
            printf("%d ",hang[i]);
        printf("\n");
        /**/
        lie=(int*)realloc(lie,(hang[0])*sizeof(int));
        for(k=0;k<hang[0];k++)
        {
            lie[k]=0;
            for(i=0;i<=j+mohang-1;i++)
            {
                if(hang[i]-k-1>=0)
                    lie[k]=lie[k]+1;
            }
        }
    }
}

```

```

    }
    /**/

    printf("按列从左到右计数:");
    for(k=0;k<hang[0];k++)
        printf("%d ",lie[k]);
    printf("\n\n");
}
if(n==0)
{
    printf("按行从上到下计数:");
    for(i=j-1;i>=0;i--)
        printf("%d ",hang[i]);
    printf("\n");
    /**/
    lie=(int*)realloc(lie,(hang[0])*sizeof(int));
    for(k=0;k<hang[0];k++)
    {
        lie[k]=0;
        for(i=0;i<=j-1;i++)
        {
            if(hang[i]-k-1>=0)
                lie[k]=lie[k]+1;
        }
    }
    /**/
    printf("按列从左到右计数:");
    for(k=0;k<hang[0];k++)
        printf("%d ",lie[k]);

```

```
        printf("\n\n");
    }
}
}
void main()
{
    int m,n;

    printf("请像这样输入球的个数和筒宽,之间以逗号隔开'm,n':");

    scanf("%d,%d",&m,&n);

    fun(m,min(m,n));

    printf("\n 如上所示,共有%d 组解的组合。 \n\n",count);
}
```

三.Faà di Bruno's Formula 之遍历 m_i 的

组合方式

1.程序：找各 m_i 组合的非人工手段

(vc++6.0, 53 行, 2 页 word 文档, 未注明各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
```

```
int *m=(int*)malloc(sizeof(int)),nn,count=0;
void get(int n,int leftovers)
{
    if(leftovers==0)
    {
        int i;
        for(i=0;i<n;i++)
            m[i]=0;
        for(i=0;i<nn;i++)
            printf("%d ",m[i]);
        printf("\n");
        count=count+1;
    }
    else if(n>leftovers)
        get(n-1,leftovers);
    else
    {
        if(n==1)
        {
            m[0]=leftovers;
            for(int i=0;i<nn;i++)
                printf("%d ",m[i]);
            printf("\n");
            count=count+1;
        }
        else
            for(int j=(leftovers-leftovers%n)/n;j>=0;j--)
            {
                m[n-1]=j;
```

```

        get(n-1,leftovers-n*j);
    }
}
}
void main()
{
    int n,i;
    printf("请输入 n 值:");
    scanf("%d",&n);
    nn=n;
    m=(int*)realloc(m,n*sizeof(int));
    printf("\n");
    for(i=0;i<nn;i++)
        printf("%d ",i+1);
    printf("%d ",nn);
    printf("\n\n");
    get(n,n);
    printf("\n");
    printf("如上所示,共有%d 组解",count);
}

```

更一般地，这种方法可以求 n 不等于 $m(\text{imax})$ 时的各项系数均为正的不定方程的非负整数解，只需要调整刚输入时的 `leftovers` 即可(不让它等于 n)，并且最好来说，你得保证输入的 `leftovers` 比 $m(\text{imax})$ 大，不然整个方程相当于系数为 $m(\text{imax})$ 的项的非系数部分恒=0 而因此系数为 $m(\text{imax})$ 的项没有用了。

再一般一点，此方法也可以求 $m(\text{imin})=m_1=1$ ，且 $m(i \neq 1)$ =任意赋值后，的各项系数均为正的多元一次不定方程的非负整数解们。当然，所需要做的只需要把刚开始包含 n 和所有 m_i 从外部输入值的顺序进行从序数小的 m_i 到序数大的 m_i 到 n =从小到大排列，取最大的一项为 n 【我不喜欢有任何 m_i 大于 n ，所以这个 n 可能不是人为能规定的，但你只需要心里想着你输入的最大的为 n ，而不是你输入的最后一个值赋值给 n ，就可以做到 n 能够被人为规定。】。另外的，如果没有输入 1 的话，我会自

动令让程序生成一个 1，这样会导致多元一次不定方程的元数(自变量数)可能也不会受人为规定，即可能不是你所想的那样，**而是可能会多一个系数为 1 的自变量**。【这是为了保证语句修改不大，且方程恒有解。当然如果你能够注意到我所说的，那么整个流程可以符合你自己的想法的。虽然有条条框框...】

更一般一点，我们可以在之前的基础上去掉 $m_1=1$ 的限制，这样就变为了“任意**各项系数均为正的**多元一次不定方程是否有解，是否有非负整数解，以及如果是，是否有有限个，有多少个，都是些什么样的组合”的问题，这样就比较 general 了。当然，程序得做出较大的修改；and 我会做出较大的牺牲(*_*)。

2.程序：各项系数均正的多元一次不定方程的非负整数解

i.既然用户都是想象和设定中的“那么懒”和“这么笨”，我还是老老实实的遵循用户的要求吧：以下便是**各项系数均为正的多元一次不定方程的非负整数解的求法**：(vc++6.0, 133 行, 6 页 word 文档，注明了各关键语句的含义。[复制粘贴即可使用](#)。)

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int *m=(int*)malloc(sizeof(int)),countk=0,count1,*x=(int*)malloc(sizeof(int));
int maxjoint(int count)                                /*构造函数 maxjoint 调取一串整数,
返回它们的最大公因数数值*/
{
    int j,i,t,*pp=(int*)malloc(count*sizeof(int));
    for(j=count-1;j>0;j--)                            /*刚开始就调换好所有相邻数据
的大小顺序以便之后的辗转相除*/
    {
        for(i=0;i<j;i++)
        {
            if(m[i]>m[i+1])
            {
                t=m[i];
```

```

        m[i]=m[i+1];
        m[i+1]=t;
    }
}
}

for(i=0;i<count;i++)                /*初始化 m 的复制品*/
    pp[i]=m[i];

    for(i=0;i<=count-2;i++)          /*由于之前是从左往右==从
小到大,那么此时得从前往后轮换*/
    {
        do
        {
            t=pp[i+1]%pp[i];
            pp[i+1]=pp[i];
            pp[i]=t;
        }

        while (t!=0);                /*跳出循环后 m[i]就是 m[i]和
m[i+1]的最大公因,而剩下的所有右边的 m[i+1]会全是 0*/

    }

    t=pp[count-2+1];

    free(pp);                        /*不用的早点释放掉*/

    pp=NULL;

    return t;
}

void get(int n,int leftovers)
{
    if(leftovers==0)
    {

        int i;

```



```

for(i=0;i<n;i++)
    x[i]=0;
for(i=0;i<count1;i++)
    printf("%d ",x[i]);
printf("\n");
countk=countk+1;
}

```

```

else if(n==1)                                     /*这个东西逻辑上最好在

```

leftovers==0 的后面,但事实上我们的算法允许它们交换位置.因为可能当 $n=2$ 的时候,进入该函数的 else 里后,发现 $x[1]$ 取 j 的最大值时,余数 $\text{leftovers}-m[1]*j=0$,这样会出现进入下一层函数 $\text{get}(1,0)$,此时我们希望的是先判断 leftovers 是否等于 0,因为已经找到一个合理解了.而如果此时先判断 $n=1$,由于 $\text{leftovers}=0$,所以条件 $\text{leftovers}\%m[0]=0$ 恒为真,使得 $x[0]=\text{leftovers}/m[0]$;语句达到和 $\text{for}(i=0;i<n;i++)$ $x[i]=0$;语句相同的初始化 $x[0]$ 为 0 的效果,并且其他之后剩下的语句均一样.这样以另一个判断标准判断其为有效非负整数解组,让人感到有点 odd,奇怪.但从逻辑上似乎也说得通,所以在我们的算法上是行得通的.*/

```

{
    if(leftovers%m[0]==0)
    {
        x[0]=leftovers/m[0];
        for(int i=0;i<count1;i++)
            printf("%d ",x[i]);
        printf("\n");
        countk=countk+1;
    }
    else

```

```

;                                                     /*什么都不做.至于为什么第

```

一个空 x_0 不必要每次都赋值为 0,是因为如果 1.在此之前有 $x_0=0$ 的整数解,那么 x_0 会被赋值为 0|2.如果在此的时候才有 $x_0=\text{某某某}$ 的整数解,那么 x_0 直接被赋值为某某某就行了|3.如果在此的时候仍没找到整数解,那么不输出就行了,保留上一个 x_0 值没有任何问题,反正下一个整数解出来的时候,要么 $x_0=0$,输出;要么 $x_0=\text{某某某}$,然后输出*/

```

    }
    else if(m[n-1]>leftovers)

        get(n-1,leftovers);                /*这个要写在 n==1 的后面,
即得先判断是否 n 减来==1.所以上次的只是特殊系数的特殊算法.因为如果出现了当
n==1 的时候 m[0]>leftovers(不知道你输入的各项系数中的最小值 m[0]是多少,而
leftovers 有可能=1,所以有可能当 n==1 的时候却进入了 m[0]>leftovers 的流程),则会
进入下一层函数,出现 m[-1],访问的空间将不在范围内.此时理应进入 n==1 的判断的.|
而上一个之所以可行是因为 m[0]恒=1,而 leftovers 不会小于 1(余数难道有小余 1 的?),
所以会直接不满足 m[0]>leftovers,然后进入 n==1 的流程.*/

        else

            for(int j=(leftovers-leftovers%m[n-1])/m[n-1];j>=0;j--) /*满足 leftovers 不等
于 0 且 n 不等于 1 且 m[n-1]不大于 leftovers 的才是可继续分解的.*/

            {

                x[n-1]=j;

                get(n-1,leftovers-m[n-1]*j);

            }
    }
int main()
{

    char s,*p1=(char*)malloc(1);

    int *p2=(int*)malloc(sizeof(int)),count2=0,sum,i;

    printf("请像这样(请不要输入非整数和 0 和负整数)输入多元一次不定方程
a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

    s=getchar();                            /*创建一个 scanf 函数用于实现我
的功能,刚开始就需要值进来,所以不用 dowhile*/

    while(s!='\n')

    {

        if(s<=57&&s>=48)

        {

```

```

count1++;                                /*统计输入了多少个"
假·数字"进来*/

*(p1+count1-1)=s;                        /*将 s 的值放入距离
p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

p1=(char*)realloc(p1,count1+1);          /*将 p1 的空间扩
张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/

}

else if(s=='-')                          /*即使输入了负数,也会被默认
为正数*/

{

    sum=0;
/*****初始化 sum 值,为了
下面的循环*/

    for(i=0;i<=count1-1;i++)

        sum=sum+*(p1+i)-'0')*(int)pow(10,count1-1-i);

    count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

    count2++;                            /*统计输入了多少个"
真·数字"进来*/

    *(p2+count2-1)=sum;                  /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

    p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

}

s=getchar();

}

if(count2-1<2)

{

```

```

printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

return 0;

}

m=(int*)realloc(m,(count2-1)*sizeof(int));

for(i=0;i<count2-1;i++)                /*初始化 p2 的复制品*/

    m[i]=p2[i];

if(p2[count2-1]%maxjoint(count2-1)!=0)
{

    printf("您输入的多元一次不定方程无整数解。 \n");

    free(p2);

    p2=NULL;

    free(p1);

    p1=NULL;

    return 0;

}

count1=count2-1;

for(i=count2-2;i>=0;i--)                /*把大于 n 的所有后面的项给
删掉*/

    if(m[i]>p2[count2-1])

    {

        m=(int*)realloc(m,i*sizeof(int));

        count1=i;                        /*记录最后的 m 中的空格
数*/

    }

x=(int*)realloc(x,count1*sizeof(int));

for(i=0;i<count1;i++)

    x[i]=0;                            /*上一个程序不需要初始化 xi
也可以,因为由于 m(imax)=n,在第一次执行函数 get 的时候,由于 n 能整除 m(imax)=n,

```

使得 leftovers 带进下一个 get 函数中,会使得之前的各个 x_i 均=0;但是这一个程序没有这个优越性,所以得初始化值*/

```
printf("\n");
for(i=0;i<count1;i++)
    printf("%d ",m[i]);
printf("%d ",p2[count2-1]);
printf("\n\n");
get(count1,p2[count2-1]);
printf("\n");
printf("如上所示,共有%d 组解",countk);
}
```

在以上的程序中,如果输入“37,48,26,49”,会发现没有提示“您输入的多元一次不定方程无整数解。”,但提示了“如上所示,共有 0 组解”。这就表明:49 整除三个系数的最大公因数 1,但是仍然没有全非负的整数解们。说明即使各项系数均为正的多元一次不定方程有整数解,也可能没有全非负的整数解。你也可以通过将这些数字“37,48,26,49”带入“求多元一次不定方程的通解”的程序中,验证一下这个多元一次不定方程是否有整数解,但对于任意整数 t_1 和 t_2 ,均没有非负整数解组。

另一项性质:如果各项系数均为正的多元一次不定方程有非负的整数解组,那么这些非负的整数解组的个数是有限的【并可以通过有限个步骤得到它们】。

ii.通过这个比较宽松条件的程序研发,我们可以进一步想一想是否能让条件更为宽松:即如果去掉各项系数均为正的限制,那么可否有非负的整数解组,其个数是有限的吗?我们可以这样想:将负系数项移到方程的 n 所在的一边,如果负系数项只有一个,那么只需要为负系数项的非系数部分 x_i 赋值为 $=(\text{所有正系数的最大公因数})/(\text{此负系数 } m_i \text{ 和所有正系数的最大公因数}) \times i$ 【其中 $i=0$ 到 $+\infty$, $i \in \mathbb{Z}$ 】,并将各个 i 下的 $n-m_i \cdot x_i$ 视为新的 n 值带入程序进行计算,便可遍历得到各个“ n ”值下的对应的非负的整数解组。从这里来看,似乎只要有一个负系数项,那么非负的整数解组数可能有无穷多组。

同理,如果有多个负系数项,那么将它们全部移到 n 所在的方程的那一边,并且各自的 x_i 均赋值为 $=(\text{所有正系数的最大公因数})/(\text{对应的负系数 } m_i \text{ 和所有正系数的最大公因数}) \times i$,【其中 i 均为 0 到 $+\infty$, $i \in \mathbb{Z}$ 】。

所以我觉得，对于有负系数项，且有正数项的多元一次不定方程，一定有非负的整数解组，且非负的整数解组的组数为无穷大。【我不再在此写对应的程序了，有兴趣可以设定各个 i 统一的取值上限，在各个有关各个 i 的排列组合下[若所有 i 的和从 0 取到负系数项的个数*imax，用这种标准来取对应的“ i 之和”的排列组合的话，会像极了“多项式定理”的有关内容。]，得到一组组新 n 值下的非负的整数解组。来看看是不是我们所想象的那样。】

四.Zengrams:幻觉艺术

【这一部分相对而言文字较多，比较“温柔”一点】

I.引入

前一部分的部分内容会被应用于这个项目，但由于每一个领域都如此精彩，以至于我现在不知道前一部分是 by-product，还是这一部分是“副产物”了：

主要内容：这与你的细心程度和你的生活经验有关：

假设有两把“齿距 \min =最小分度值”的梳子，它们的最小间距/最小正周期(一般)互异。在这样的简单条件下，我们把问题以这样的形式提出：

- 1.我们将两把梳子开口对开口，各梳子所在的平面相互平行，且上下错位地，沿着平面方向，相互靠近，并尝试遮盖彼此。做到之后，再将其中任意一个或者你拿两个也行：使得它们具有相对运动速度，并假使 $V_{\text{相}}$ =定值某，即以某匀速进行相对运动。
 - 2.在运动过程中，当 A 梳子的齿与齿之间的空白与 B 梳子的白空处相互重合时，即当 2 个梳子的可透射光的间隙相互重叠时，由于任意一个梳子的空白都没被对方那只梳子的“黑刻度线”：齿子，给挡住，因而你会看到“闪亮”一次。【即相对于黑对白、白对黑、黑对黑来说，白对白对你来说具有与众不同的抓人眼球的特点。】
- ①.由于最小分度值，一般 A 尺与 B 尺并不相同，所以一般情况下你无法同时既观察黑对黑又观察白对白——原因是，当我们选择了某个对象(比如白)作为观察对象后，我们会勒令“A 尺的每一段白间隔的长度=B 尺的每一段白间隔的长度”，此时由于“最

小分度值=相邻亮条纹中心距=相邻暗条纹中心距= $2*(黑/2)+白=2*(白/2)+黑=白+黑$
 对于 A 尺与 B 尺来说一般互异，因而当白 A=白 B 时，由于“ $d_{Amin} \neq d_{Bmin}$ ”导致

“白 A+黑 A \neq 白 B+黑 B”进而将导致黑 A \neq 黑 B。若我们在黑 A \neq 黑 B 的时候去观察黑对黑，将发现 A 尺的任意一个黑段没有任何一次机会能完全与 B 尺的任意一段黑完全重合。——注：“完全重合”是我们判断“相遇”的标志，“能完全重合”使我们的研究对象区别于其他无关变量的标志。

由于①所带来的困扰，我们尝试着进一步抽象出②：本质上我们仍可以看黑 A \neq 黑 B 时的黑对黑，它的“相遇规则”不再是“完全重合”，而是黑段中心线与黑段中心线重合时即为一次相遇。同理其实白段与白段的“完全重合”其本质上也是白段中心与白段中心的重合(但我们的眼睛刚开始甚至到现在可能都可不是这么想的哦)。并且，可以证明，各尺子上的相邻的黑段中心线与白段中心线之间的间距=任意相邻两线的间距=(白段+黑段)/2=原最小分度值的 1/2=新的最小分度值。所以我们现在开始把模型从“梳子”进一步抽象成“尺子”；把“两把梳子”想象成“游标卡尺的主尺和游标”等等(即：把段与段的重合抽象成线与线的相遇，黑线虽仍有宽度，但我们已不再计其宽度了)。

②.为了让问题更本质地呈现在读者面前，现在研究对象同样也可被表述为：现在两把梳子变成了两把尺子，A 尺的黑线与 B 尺的黑刻度线重合时，在重合的位置有一次“闪亮”，并且是任意的 A 尺的刻度线与任意的 B 尺的刻度线有任何一次相遇(即重合)时，均会在相遇地点闪亮一次。【注：白对白=黑对黑+闪光 \rightarrow →花费最低能量，“自动”筛选出这些稀有信息；黑对黑+不闪光=黑对白=白对黑 \rightarrow →选择性略过这些不稀有的信息。

3.问题来了：i.当“主尺和游标的刻度线在任何时间任何地方重合时便闪光一次”这种宏观的、密集的闪光集合的元素，在“以匀速进行相对运动”的条件下。有什么规律可循？闪光与闪光的空间距离与闪光的函数关系？闪光与闪光的时间距离与闪光的函数关系？闪光与闪光的空间距离与时间距离的函数关系？ii.先提出其中的一种规律，不然这个问题无法继续似的：目之所能及时，你将看见有一种“等间距的闪光点们”，以当前最快速度($>V_{相}$)向某个方向做“匀速运动”(相对于其他“可以有速度的闪光点们”)，问这个速度关于“ $V_{相}$ 、尺 A 的最小分度值 a、尺 B 的最小分度值 b”的函数关系？

II.理论解释(part 01)

下面我开始连起来解释 3.中的 i.与 ii.:

人类这个高度有序，但大脑却处于混沌状态的矛盾的杰作，总会偏向于集中注意力于那些与之相似的现象：我们渴望规律，一小部分是因为我们渴望安全感，另一大部分却是：这个世界“能被找到规律”的，我们身边的现象，相对于“没有规律”和“有规律，但暂时没法找到它”的现象，太少了。以至于我们总是倾向于去寻找一种类己的存在感：这些对于自己因高度相似而熟悉，但对整个宇宙来说因稀少而十分陌生的存在。

这就像规则一样，不仅统治着我们的“条件反射”层面，同时也存在于我们的“非条件反射”层面——意识层面。所以不管你愿不愿意同意它，屈服于它，你终究会，并且愿意同意它、屈服于它的；连起来说就是：你会愿意的。

这次的例子中，就会出现：“你总会看到你愿意看到的，这句话总是对的，以至于甚至那些你所不愿看到的，一旦你看到了，立马就将成为你愿意看到的，并且自此，自一开始它就是你愿意看到的。”这句话，不如说这个思想，要想从“矛盾”变成“看似矛盾”，一种让它成立的解释便是：“你不完全是你”。

现在我们回到这个例子中：对于非闪光的 majority，我们倾向于注视“闪光点——这些 minority 们”；而在这些“同样是闪光点”的闪光点中，对于集合中杂乱闪光的元素 majority 们，我们又更倾向于注视“有着相对更有规律/规律体现得更明显的 minority 们”。

综上，人眼自身的存在，导致人类所看见的真实，被自身的局限所局限在了过滤了杂音之后的图景上，（虽然这种局限或许是种 gift，上帝作为这些闪光点，更愿意让我们给捕捉到，以至于强迫赋予了我们能力，并强制我们无时无刻不使用它。）

这时，“相对更有规律”便体现在“均匀时间”和“均匀空间”上——即“ $\Delta t/\text{闪}$ ”为某一常值和“ $\Delta x/\text{闪}$ ”为某一常值的这两个基础幻想上；又由于恰好在这元素中，有那么些元素在那么些时刻和那么些空间位置上同时满足用有这两种特性的特性，于是我们便一步步地缩小了 focus 范围，现在便落在了 $(\Delta x/\text{闪})/(\Delta t/\text{闪}) = \Delta x/\Delta t$ 为某一常值的闪光集合上；现在又由于 $\Delta x/\Delta t = v$ 为常值的元素集合又有几大批，它们又组成了一个新的分类标准下的新集合；所以我们进一步，又尝试着搜寻新集合中最易搜寻的特点凸出的元素：即 v 为常值的元素中， v 最大的元素。

关于为什么我们要研究 V_{\max} 的闪光元素，我们来进一步更细节地探讨它：通过先提出一个问题：有没有这样一些时刻，A 尺的任意刻度线与 B 尺的任意刻度线不重合？换句话说，是否每个时刻都总有 A 尺的 somewhere 的刻度线对齐于 B 尺的这里的刻度线？（后者，绿色的）答案是否定的：我们假想有一个“有刻度线对齐的”时刻，那么此时总有一对 A 尺的刻度线与 B 尺的刻度线，在各个相邻的刻度线中，其间距

是 >0 但最接近0的；也就是说，在各对相邻刻度线中，总有一对刻度线间距最小，为 Δx_{\min} ，并且这对刻度线总是：一根属于A尺，另一根属于B尺。【注：它所属的集合：“各对相邻刻度线”，包括了“同一根尺子(并非两尺都可，而是 $\min(a,b)$ 所对应的那个尺子A或B)的一些对相邻刻度线”以及“A尺和B尺的一些相邻刻度线”。】

讨论 V_{\max} 的时机成熟了：

(1).不存在所有刻度于A和于B都被对齐的时刻。【至少有一个尺子的部分刻度没有对应的对面的尺子的刻度来对齐，但可以有一个尺子的刻度完全被对齐的时刻。】

(2).存在一些时刻，于A和于B，所有刻度均找不到“对齐/配对”的另一半。

第二点正是我们想讨论的：这些时刻就 lies in Δx_{\min} 之中有刻度划过/存在的时候：那么在这些时候，你的眼睛都看到了什么？是的，你看到了“看不到闪光出现的时刻”的存在，或者更准确的说，是“看不到闪光的時刻的连续集合”；是“看不到闪光的时段”——which 正是让闪光成为闪光的东西：通过衬托。

所以你知道了：其实闪光不是持续发生的；其实在闪光的片刻之外，充斥着远多于闪光的片刻们；其实闪光还是很稀有的；其实闪光之所以称为闪光，就是因为它是相对的时刻而不是时段，并且它相对于非闪光是少数者们。

这样一来，我的引入便可以以一种更和谐的方式提出：那么在这些时刻，你都在干什么？读者别笑，这个问题问得恰到好处：GOTO:没错，你在等待着下一次闪光的出现！并且，你会，你总会，有意观察，它间隔多久出现，它在哪里出现，即它间隔多远出现，它以什么样的形式出现——由于你的脑子可以存储数据，在缓冲区，你又总是把每一次它的出现(的各个属性)，与上一次它的出现作比较；由于你的脑子还够用，你又总是把“这次它的出现与上一次它的出现所比较得出的结果”与“你上一次进行的：上一次它的出现与上上一次它的出现比较得出的结果”比较，又得出了新的结果。

这就是“距离与时间间隔进化于 single 空间点和时间点”；速度进化于空间间隔和时间间隔；加速度进化于速度；加加速度进化于加速度……人类没事就把已有的资料搜集起来(人类手头只有这么多东西，但似乎已足够？)，逐层深入的进行自我纵向横向对比，一遍一遍的翻阅旧资料，一次次地得出新结论——这就是人类原始但又先进的，却又是唯一的，发现真理的手段——模仿游戏；这就是人类在用规律来找规律——所有当下的 disorders，终究会在某个思维维度以某种该维度的 order 出现，这就是人类的智慧/信仰/执念；这就是人类。

不过你不会苟同最后那个说法，因为人类似乎远比“这”更复杂。

III.理论解释(part 02)

我们回到 GOTO 标记处, 在你等待下一次闪光出现的时候, 你会想: 它将什么时候出现呢? 换句话说, 你在想: “接下来第一次出现的闪光, 将于哪个最临近的时刻出现呢?” ——当你身处无光的 moments 所构成的涓涓细流中时, 你最关心的是: where's the next moment of sparkling in the time arrow? 而不是 “where's the next sparkling's next moment of sparkling in the time arrow”。也就是说, 只有紧邻的、第一个, 才对你, 才对这时候的你, 有意义。而之所以第二个对当下的你没有意义是因为: 一旦该无光时刻 t_0 接下来的闪光第一次发生后, 这时 t_0 已经走到了 t_1 , 而对于 t_0 的第二次闪光, 现在对于 t_1 而言又晋升为了第一次闪光了。

由于你无时无刻不处在“当下”, 那么现身处 t_1 的你, 吸取了之前 t_0 的教训, 已不会再贪恋“第二次”第二次了: 因为你已错过了对于 t_0 的第一次, 若你在 t_1 时又去想念相对于 t_1 的“第二次”, 那么紧接着你又会错过 t_1 的“第一次”, 并且同时你也错过了相对于 t_0 的“第二次”。那么你将脸红而无颜/言面对 t_0 : 你错失了它给你预留的两次美景。

记住, 每一次闪光都暗示你, 过去的已经过去, 且未来只局限于下一次闪光处为止, 而不是那之后。并且属于你的现在的, 只有这相邻两次闪光之间, 这段时间。也就是说, 每一次闪光后, “这一次闪光后的第一次闪光”便不能再称为“这个闪光前一次闪光的第二次闪光了”。因为连同这次闪光, 上一次闪光也已经被时间给抹去了, 无法再作为你的时间原点了, 或者说, 已无法再作为你的新的时间原点了。

我们生活在一个紧接着下一个的闪光中, 我们生活在一个紧接着下一个的“(一个紧接着下一个的闪光的)间隙”中; 这个最小时间间隔 T_{min} 与 Δx_{min} 有关, 那么, 在此刻闪光之后, 到紧接着的下一闪光之时(前), 这段时间长 T_{min} 为多少? 在之前的铺垫下(P56), 可见 T_{min} 直接对应 Δx_{min} , 即对应“当有刻度对齐的时候的相邻刻度间的最小间距”。其理由不言而喻: 当这次对齐结束后一段时间内, 由于 Δx_{min} 已经是 Δx 中的 min 了, 所以在 Δx_{min} 没有被走完之前, 不存在任意 Δx 被走完, 即这段时间内不会有闪光发生。在保证了“ Δx_{min} 对应的 T_{min} 内无闪光, 以及 Δx_{min} 对应的 T_{min} 时/后有闪光”之后, 我们便得知它们——对应且有 $T_{min} = \Delta x_{min} / V$ 相。

而根据 $V = \Delta x / \Delta t = (\Delta x / \text{闪}) / (\Delta t / \text{闪})$, 我们知道, $\Delta x = k(a \text{ 或 } b)$, 即任何一次闪光都闪在被观察的尺子的某个所对应的刻度上, 而不是其他地方。因而闪光与闪光之间的空间距离 Δx 是被观察的尺子的最小分度值 $(a \text{ 或 } b)$ 的整数倍。接下来我们就有

$V_{\max} = \max(\Delta x / \text{闪}) / \min(\Delta t / \text{闪}) = (\Delta x_{\max} / \text{闪}) / (\Delta t_{\min} / \text{闪}) = \Delta x_{\max} / T_{\min} = \Delta x_{\max} / \Delta x_{\min}$
 * V 相, 现在仅剩的工作便是寻找这个 Δx_{\max} 以及这个 Δx_{\min} , 由于这个 Δx_{\max} 仍
 $= k(a \text{ 或 } b)$, 所以接下来的工作便是: 寻找 k 和 Δx_{\min} 。

我们接下来会在寻找 Δx_{\min} 的路上找到 k , 因为它们同时对应于 “以 T_{\min} 为间隔的两次时间上相邻的闪光” —— 同一个事件之 “两次有刻度对齐的时刻”。假设尺子 A 和尺子 B 都无限长, 那么无论它们的最小分度值 a 和 b 分别具体取值为多少 (在 Q 中而非 R 中取), 当 “有刻度对齐时”, 一定 “有无限个刻度对齐”, 且它们的空间分布是均匀的; 并且很容易理解这时, 甲. 相邻的 “对齐的刻度” 的间距长度均为 $[a, b]$, 即 a, b 的最小公倍数; 乙. 相邻的 “没对齐的刻度” 的间距长度的最小值 $\Delta x_{\min} = (a, b)$ 。

这两个结论可以用圆向量来察看它们的正确性: 在 “有刻度对齐” 的地方, 挑选其为圆向量的公共起点; 将 A 或 B 的最小分度值 a 或 b 作为圆向量的基础大圆, 选另一个作为 “步子” 进行跳跃; 可知在 “步子” 跳跃 “ $[a, b] / \text{步长}$ ” 那么多次之内至少有一次离原点最近, 且距离 $= (a, b)$, 且实际上至少有两次距离原点 (a, b) , 并且这两个落脚点分居原点两侧。更多地还有: i. 对于任意 $< [a, b] / 2$ 的落脚点 “ $k * \text{步长}$ ”, 均有其关于 $[a, b] / 2$ 对称的对应的镜像: “ $[a, b] - k * \text{步长}$ ”。【这是因为跳跃的时候可以顺时针也可以逆时针, 跳过去还可跳回来, 而它们的所有落脚点的最终分布情况是一样的。举个生动的例子: 设有 $19x - 7y = 1$, 一个解为 $(x, y) = (3, 8)$, 则一定还有一个解 $(7-3, 19-8)$ 使得 $19x - 7y = -1$, 或使得 $7y - 19x = 1$ 。→→→更一般点地, $ax_0 - by_0 = (a, b)$, $([a, b] - by_0) - ([a, b] - ax_0) = (a, b)$, $b([a, b] / b - y_0) - a([a, b] / a - x_0) = (a, b)$, $a(x_0 - b / (a, b)) - b(y_0 - a / (a, b)) = (a, b)$ →→→在接下来的最后一步应用这个小东西 (其实这是通解公式中 $k=1 \text{ 或 } -1$ 时的情景), 有 $ax_0 - by_0 = (a, b)$, $ax_0 - b(y_0 + 1) = (a, b) - b$, $a(-x_0) - b(-y_0 - 1) = b - (a, b)$, $a(b / (a, b) - x_0) - b(a / (a, b) - y_0 - 1) = b - (a, b)$ 。这种简单的常规操作可以灵活地应用于其他 c 值上。】ii. 直线上 “空间上相邻的对齐的刻度” 的间距 = 对应的圆向量中 “时间上相邻的又一次回到原点所经历的几个大周长 * 大周长长度 or 几个步子 * 步长” $= [a, b]$ iii. 直线上 “相邻的没对齐的刻度的间距” 的最小值 $\Delta x_{\min} =$ 圆周上距离原点最近的落脚点, 与原点的距离 $= (a, b)$ 。

解决了 $\Delta x_{\min} = (a, b)$ 后, 我们来继续看看 k 如何呈现: 同样与 Δx_{\min} 对应的, T_{\min} 表示与 Δx_{\min} 对应的时间间隔, $k(a \text{ 或 } b)$ 表示与这种空间间隔 Δx_{\min} 所对应的另一种含义下的空间间隔。——我们像或不像以前注意闪与闪之间的时间间隔问题一样, 现在开始注意一个闪与紧接着的下一个闪之间的, 空间间隔问题: 继之前的某个被你的目光所选中的刻度对齐之后, 在某次闪光之后, 在接下来的一段静寂之后, 在你聚精会神之后, |*下一次它在哪闪呢*|?

矛盾的是，下一次闪亮的不仅是“它”，而是“它们”。即紧接着的下一次闪亮，闪亮的是一排以[a,b]为间隔均匀分布的闪光点，正如刚刚过去的那次闪光一样。这样一来，被“|**|”所标记的语句的问法便有点让人奇怪了：正如上次那次闪光一样，每一次都是一群重合点在闪光，然而似乎连同上次，每次我们的关注对象却都仅仅局限/聚焦于其中的“某一个闪光”？——这是因为，即使你动用了你的余光和大脑剩下的cpu，去尝试着“get a whole picture of”那一排闪光，你的视觉和意识焦点仍只有那一个，不是吗？——所以，在余光允许了你知晓了其他闪光的存在后，你仍然会选择注意那个你原先所注意的那个闪光附近的闪光：

在短短的 T_{min} 内，你的视野中央连续发生了以下故事/你也将因此发生以下思考——“我所注意的A尺的那个刻度，与B尺的刻度，于此刻，对齐了，闪光了。”——“V相使它们分离了，熄灭了”——“接下来是一片空白期 or 黑暗期，此时我在极力搜寻可能发生的下次闪光的空间位置，并在时间上等待着它”——“但由于此 T_{min} 太短了，我无法迅速抽身离开我的眼睛之前注意的位置，或者我走不了多远，并且由于万籁俱寂，我不知道朝哪里走”——“朝哪儿走都是徒劳的，所以在内在和外存在的双重条件约束下，我的眼睛连同我的意识，都还停留在之前闪光的地方周围”——“其他地方尚未发生任何事件，我便因此不会反应；因为我总是|因此而反应|，现在没有任何|此|，所以理所应当不反应。”——“闪光了！而且就在我左右！这也恰好是我的视觉焦点暂留的、以前闪光的地方，它的左边以及右边都闪了光！”——“等我刚用余光看到无限远处也有闪光时，意识的准心所刚移动到的，左 or 右的闪光点，又熄灭了，连同无限远处的灯光。”——“等我回过神来，在新的焦点处的左边和右边又有灯光了。”——“连起来便是：原先焦点左右两边的灯光吸引我把眼球焦点移向其中一方，等到或还未等到新焦点到达其中一个地方，这个地方的灯便熄灭了，‘同时’，这个地方的左右又亮起了新的闪光。”——于是，循环(单个循环体的循环)已经开始，或，循环(一个完整的循环体)已经完成。

1.你的眼睛，catches, falls behind, 一直交替和循环，惯性作用下你的意识焦点会沿着初始移动方向一直移动下去，所以你看到的一直是你刚开始时所以为看到的景象：一排闪光们因你视觉焦点的向右移动而→→→，由于你不是像我一样知道闪光是因你注意力的右移而右移的，所以你仅仅因不愿你的目光跟不上你所认为的闪光的脚步，而一直在向右移动你的注意力，所以你看到的一直是你自己意识的具象。——这就是“你所看到的都是你想看到的，即使是没想过能看到的、不想看到的，也都是曾经并正，希望看到的”的具体阐述和典型例子。

2.但或许有那么一些瞬间，你发现你的灯也时不时的像你一样(- -)临时改变主意，在往左←←走。如果你自始至终只改变一次注意力的方向，这一排闪光或许从你改变注意力开始就像你的注意力一样改为执着地一直向左走了。不过通常等你刚这样

想时，眼睛和闪光又不听使唤了：它怎么还 $\leftarrow\leftarrow\rightarrow\rightarrow$ 左右跳动呢？——不仅对于单个闪光是这样，同时你还发现整排闪光都在 $\leftarrow\leftarrow\leftarrow$ 、 $\rightarrow\rightarrow\rightarrow$ 左右交替跳动，这是幻觉吗？如果不是，它们的行动模式一旦设定怎么可能改变？也不可能初始设定成这样呀？！

3.确实，“它们的行动模式”并没改变；而是“你的思考/认知模式”一直在改变——正如你越不明白就越要去想，你朝一个方向想不通就会不停的换角度思考，导致闪光和你的思想一齐变换；然后你更想不通，更想想，它便更要让你迷惑不解。

恭喜，另一个循环又已经启动或已经完成。

IV.理论解释(part 03)

在弄清楚“你弄不清楚”这件事后，讨论 k 和 V_{\max} 的解的时机，便已经成熟。要声明的是，我们的所讨论的 V 值，只有仅仅当你在对闪光进行“连续的同方向幻想”时，讨论它才较有意义；不然就会出现之前的第 2.点中后面所提及的：这种有方向变化的，即有正负号改变的“速度”，这虽然也是可用总位移 $\Delta x/\Delta t$ 量度的，但这里我们不讨论它。||现假设之前的第 1.点和第 2.点中出现的，向左和向右的速度，分别对应(但不管谁对应谁) $V1$ 和 $V2$ ，我们的 V_{\max} 便= $\max(V1,V2)$ ，同时与之对应的也会有个 $V_{\min}=\min(V1,V2)$ 。【当然，除了这种意义下的 V_{\max} 外，还有更大的 V ，但如果这样的话， V 会因没有上限而没有最大值；同样除了这种意义下的 V_{\min} ，也还有更小的 V (你可以挑选更长时间间隔的两次一对闪光)，但如果这样的话， V 会因在大于 0 的区间内没有下限而没有最小值。】

此时， $V1$ 和 $V2$ 会分别对应自己的一个 $k1$ 和 $k2$ ，且有 $k1=x1-x0$ ， $k2=x2-x0$ ，即 $k1$ 、 $k2$ 分别是被观察的尺子中，以之前你所曾在意的一个闪光为零刻度 $x0$ ，刚刚同时在它左右闪光了的两个闪光的相对坐标。并且有方程： $ax-by=(a,b)$ ，对应此方程，设移动的尺子叫 A 尺，最小分度值为 a ；被观察的尺子叫 B 尺，最小分度值为 b ，取它的绝对值之和最小的一对整数解 (x,y) ——则 y 即为 V_{\min} 所对应的 $\min(V1,V2)$ 所对应的 $V1$ 或 $V2$ 所对应的 $k1$ 或 $k2$ ；并且进一步地， $y - \frac{y}{|y|} \frac{[a,b]}{b}$ 即为 V_{\max} 所对应的 $\max(V1,V2)$ 所对应的 $V2$ 或 $V1$ 所对应的 $k2$ 或 $k1$ 。【其中 $\frac{y}{|y|}$ 表示取 y 的符号。】

这里这对绝对值之和最小的整数解 (x,y) ，有个非常特殊的性质：①.它们所对应的 $|ax|$ 和 $|by|$ 全都满足 $\leq [a,b]/2$ ，并且：②.只有这对绝对值之和最小的整数解 (x,y) ，才有这个性质，其他的整数解 (x,y) 所对应的 $|ax|$ 或 $|by|$ 全都 $> [a,b]/2$ 。【当然若选取新的零刻度，它们也可 $\leq [a,b]/2$ ，这仅仅是周期为 $[a,b]$ 下的一个代表而已】。所以我们

可以利用这个特性，找到一个 $|ax|$ 或者 $|by|$ 满足 $\leq [a,b]/2$ 的解 (x,y) ，那么其所对应的 $|ax|$ 和 $|by|$ 全满足 $\leq [a,b]/2$ ，并且只有这一个解 (x,y) 满足。这个规则我们等下可能会写进程序，用来找这个“最小整数解 (x,y) ”【用来判断是否(绝对值之和)“最小”】。

对于这个问题，对应地在之前，【这里默认 a, b 均正】对于 $ax-by=c$ ， $x=x_0-t*b/(a,-b)$ ， $y=y_0+t*a/(a,-b)$ ，可从中窥见不论 x_0 和 y_0 为何值， x, y 均一定至少有一个值的绝对值分别 $\leq b/(a,b)/2$ 、 $a/(a,b)/2$ ，并且也最多只有这一个。不同的表示方法可以代表同样的含义，上一段的表述中， $|ax| \leq [a,b]/2$ 等价于 $|x| \leq [a,b]/2/a = b/(a,b)/2$ ； $|by| \leq [a,b]/2$ 等价于 $|y| \leq [a,b]/2/b = a/(a,b)/2$ 。所以对于即使是 c 不等于 $+(a,b)$ 的 (x_0, y_0) ，也将有： x, y 的解的绝对值，有且最多有一个，分别 $\leq b/(a,b)/2$ 、 $a/(a,b)/2$ ，即存在且最多存在一个 (x_0, y_0) ，使得 $|x_0| \leq b/(a,b)/2$ (或 $|y_0| \leq a/(a,b)/2$)，并且若其中一个对于 (x_0, y_0) 成立，则另一个对于它也将成立。

由于在这个方程 $ax-by=(a,b)$ 中，规定 A 尺为被移动的尺子， B 尺为被观察的尺子，那么 i. 当解为双正时，表明 $ax-by=+(a,b)>0$ ，即有 $ax>by$ ，又由于双正解，则 $|ax|>|by|$ 。这表明相对于刚对齐过刻度之处的零刻度线， B 尺上的下一次闪亮点/下一次相邻的刻度对齐处，距离此零刻度线的距离 $|by|<A$ 尺上的距离这个时间上的 next 闪光点最近的刻度，所距离此零刻度线的距离 $|ax|$ 。此时 V_{min} 与运动方向 V 相反；ii. 当解为双负时，表明 $a(-x)-b(-y)=-(a,b)<0$ ，即有 $a(-x)<b(-y)$ ，又由于双负解，则 $|ax|<|by|$ 。这表明相对于刚对齐过刻度之处的零刻度线， B 尺上的下一次闪亮点/下一次相邻的刻度对齐处，距离此零刻度线的距离 $|by|>A$ 尺上的距离这个时间上的 next 闪光点最近的刻度，所距离此零刻度线的距离 $|ax|$ 。此时 V_{min} 与运动方向 V 相同。

若记 V 相为正，且记与 V 相同方向则为正。并记 $V_2=V_{min}$ ， $V_1=V_{max}$ 。那么根据以上，i. 当解为双正时， V_{min} 与 V 相方向相反，则 V_{min} 值为负。另将 $k_2=y$ 和 $\Delta x_{min}=(a,b)$ 带入之前得到的 $V_{min}=\frac{|k_2| \cdot b}{\Delta x_{min}} \cdot V$ 相，有： $V_{min}=\frac{y \cdot b}{(a,b)} \cdot V$ 相，这个是 V_{min} 的大小。加上 y 为正解且 V_{min} 值得为负，则综上，此时 $V_{min}=-\frac{y \cdot b}{(a,b)} \cdot V$ 相。ii. 当解为双负时， V_{min} 与 V 相方向相同，则 V_{min} 值为正。而其大小仍为 $V_{min}=\frac{|y| \cdot b}{(a,b)} \cdot V$ 相，考虑到 y 为负解且 V_{min} 值得为正，则综上，此时仍然有 $V_{min}=-\frac{y \cdot b}{(a,b)} \cdot V$ 相成立。

综上，不管解为双负还是双正【之前就应该说明白的，或者说本来不说也该明白的：该方程 $ax-by=(a,b)$ 要么解为双正，要么解为双负，没有其他情况】，均可由同一个式子 $V_{min}=-y \cdot \frac{b}{(a,b)} \cdot V$ 相表示 V_{min} 的大小和方向。同样的道理将 $k_1=y - \frac{y}{|y|} \frac{[a,b]}{b}$ 和 $\Delta x_{min}=(a,b)$ 带入之前得到的 $V_{max}=\frac{|k_1| \cdot b}{\Delta x_{min}} \cdot V$ 相，有： $V_{max}=\frac{|y - \frac{y}{|y|} \frac{[a,b]}{b}| \cdot b}{(a,b)} \cdot V$ 相，这个是 V_{max} 的大小。由于不管 V_{min} 的符号和解的正负性如何，均有 V_{max} 与 V_{min} 方向相反，且 $y - \frac{y}{|y|} \frac{[a,b]}{b}$ 已保证自身恒与 y 异号。则 i. 当解为双正时，由于 V_{min} 值为负，

则 V_{\max} 值为正, 且有 $V_{\max} = -(y - \frac{y}{|y|} \frac{[a,b]}{b}) \cdot \frac{b}{(a,b)} \cdot V$ 相。ii. 当解为双负时, 由于 V_{\min} 值为正, 则 V_{\max} 值为负, 且有 $V_{\max} = -(y - \frac{y}{|y|} \frac{[a,b]}{b}) \cdot \frac{b}{(a,b)} \cdot V$ 相。

综上, 任何情况下, 均有 $V_{\min} = -y \cdot \frac{b}{(a,b)} \cdot V$ 相及 $V_{\max} = -(y - \frac{y}{|y|} \frac{[a,b]}{b}) \cdot \frac{b}{(a,b)} \cdot V$ 相。现对后者进行一次化简, 并同时统一前后两者的形式, 则有 $V_{\max} = -(b \cdot y - \frac{y}{|y|} [a,b]) \cdot \frac{1}{(a,b)} \cdot V$ 相 $= \frac{\frac{y}{|y|} [a,b] - b \cdot y}{(a,b)} \cdot V$ 相; $V_{\min} = -\frac{b \cdot y}{(a,b)} \cdot V$ 相。——综上, 两个非常伟大的公式已经出炉了: $V_{\max} = \frac{\frac{y}{|y|} [a,b] - b \cdot y}{(a,b)} \cdot V$ 相, $V_{\min} = -\frac{b \cdot y}{(a,b)} \cdot V$ 相。【其中 $-b \cdot y$ 同时因 $ax-by=(a,b)$ 也可写作 $(a,b) - a \cdot x$, 你可以因此只将 V_{\min} 写作 $V_{\min} = \frac{(a,b) - a \cdot x}{(a,b)} \cdot V$ 相, which 看起来和 V_{\max} 的分子的形式 $\frac{y}{|y|} [a,b] - b \cdot y$ 非常相似。不过其实 $-b \cdot y$ 也非常棒, 所以其实没有必要这样做(况且这样做还加深了你除了求 y 之外还得用到 x 的困难)。】

V.理论解释(part 04)

接下来, 我们会应用第二个程序的前面的一小段, 以之为基础并稍加修改, 将这个实际问题的通解写成一个小程序。不过在此之前, 我还得修正一下内容: 我们不再采用 $\frac{y}{|y|}$ 作为取 y 的符号的手段, 因为可以看见, 数学语言采用 $\frac{y}{|y|}$ 是为了简洁精炼, 而计算机语言不会用 $\frac{y}{|y|}$, 一是因为它作为后台的、不可见的代码不太(若能达成目标, 还是精炼点好)要求非常精炼, 更重要的是它为了实现目的而不能采用这个表达: 当且仅当 $a=(a,b)$ 时, 会出现 $ax-by=(a,b)$ 中的 (a,b) 会被直接分配到 x 项, 而 y 项得到的系数是 0, 由此导致解得 $y_0=0$, 那么如此一来 $\frac{y_0}{|y_0|}$ 对于计算机来说就会因分母=0 而变得不可解。所以其实因为 $\frac{y_0}{|y_0|}$ 完不成目的, 其实连数学语言也不能用它的: 因而我们得修正这个表达, 使得程序能够对于一切可能的、正确的 a, b 组合, 都能输出结果且都能输出正确的结果: 解决方案便是: 立个 flag, 用 flag 替换掉 $\frac{y_0}{|y_0|}$; 且当 $y_0 < 0$ 时, $\text{flag} = -1$, 其他情况 flag 的值均为 1。以上解决方案解决了两个问题, 一个是显在问题: $\frac{y_0}{|y_0|}$ 的分母=0 时无法输出的问题; 另一个是潜在问题: 当 y_0 确实=0 时, 我们创建了对它的新的正确的解法: 此时我们认为对应的物理情景中, 各 V_{\max} 和 V_{\min} 对应于解为双正时的物理情景, 即 V_{\min} 为负, 与 V 相反向(不过不同的是, 这个情况中的 $V_{\min}=0$); V_{\max} 为正, 与 V 相同向。——这一方面是因为“ x_0 为正, $y_0=0$ ”中“有半个正数 x_0 ”; 另一主要的方面是因为, 眼睛首先会盯着原初的 A 尺移动, 因而在残存的印象的影响下和惯性所导致的眼睛的接下来的速度(方向)会不易改变于其初速度(方向)的原因, V_{\max} 会优先与 V 相同向, 即 V_{\max} 会优先选择为正, 即 flag 值为 1。

当然，说这么多与此有关的内容，我还想说的是，其实以上中也可以写成“当 $y_0 \leq 0$ 时， $\text{flag} = -1$ ”，即当 $y_0 = 0$ 时，也可以认为 $\text{flag} = -1$ ，即也可以认为解为双负，即 V_{\max} 也可以被认为负，即 V_{\max} 可以被认为与 V 相反向。因为看待事物的想法因人而异，且对于同一个人来说，也会因时而异。

另外的，与之对应的还有，当且仅当 $b = (a, b)$ 时， $ax - by = (a, b)$ 中的 (a, b) 会被直接分配到 y 项，而 x 项得到的系数是 0，由此导致解得 $x_0 = 0$ ，但此时没有任何一个 V_{\max} 或 V_{\min} 因此 $= 0$ ，所以其实这个情况不是与 $a = (a, b)$ 情况所等价的，其具体原因便是， A 尺和 B 尺的属性，一个是被移动的尺子，另一个是被观察的尺子，尺子的属性不是可逆的，因而两个情况不是等价的。

更特殊的，若 $a = (a, b)$ 的同时，也有 $b = (a, b)$ ，即若 $a|b$ 的同时，也有 $b|a$ ，那么此时 $a = b$ ，注：不会有 $V_{\max} = -V_{\min} = \pm V$ 相【主要是仅仅不会有 $V_{\max} = -V_{\min}$ 】。而是与“当且仅当 $a = (a, b)$ 时的情况”一样，“一个正解 $x_0 = 1$ ，一个 0 解 $y_0 = 0$ ”

【这是因为我在程序中设定了默认 x_0 的各阶段系数为最小值。不然就会是“一个 0 解 $x_0 = 0$ ，一个负解 $y_0 = -1$ ”，对应于 $V_{\max} = \frac{y}{|y|} \frac{[a, b] - b \cdot y}{(a, b)} \cdot V$ 相 $= 0 \cdot V$ 相； $V_{\min} = -\frac{b \cdot y}{(a, b)} \cdot V$ 相 $= V$ 相，此时显然 $|V_{\min}|$ 不是 \min 】，并且由于此时对 V_{\max} 和 V_{\min} 的方向的看法也因人而异、因时而异，所以这个情况可以被整合至“当且仅当 $a = (a, b)$ 时的情况”，形成“当 $a = (a, b)$ 时的情况”。①.但在程序中，这个“当 $a = (a, b)$ 时的情况”没有被单列出来提示读者“ V_{\max} 可以是 $\pm k \cdot V$ 相”，而是保留了“ V_{\max} 为 $+k \cdot V$ 相”的那个小小的立场。②.程序中数学思想上确实整合了“当且仅当 $a = (a, b)$ 时的情况”和“ $a = (a, b)$ 的同时， $b = (a, b)$ 的情况”为了“当 $a = (a, b)$ 时的情况”，但在写的位置上，它们仍然是分开的。这是因为我想检验这个算法下恒有“ $\text{abs}(x_0) \leq B/\text{joint}/2$ ”的同时，恰好只有“ $a = (a, b)$ 的同时， $b = (a, b)$ 的情况”不在“ $\text{abs}(x_0) \leq B/\text{joint}/2$ ”之内，所以我把“ $a = (a, b)$ 的同时， $b = (a, b)$ 的情况”写在了下面语句的下面，且把“`printf("\n 这个情况应该是不会出现的- .\n\n");`”改为了“`printf("\n 这个情况在不是 $a = b$ 即 $\%d = \%d$ 的时候,应该是不会出现的- .\n", A, B);`”。

1.程序：Zengrams 幻觉艺术

不过我现在把“ $\text{abs}(x_0) \leq B/\text{joint}/2$ ”改为了“ $\text{abs}(y_0) \leq A/\text{joint}/2$ ”，就可以还原为之前的语句(结构)了。代码如下：(vc++6.0，106 行，3.5 页 word 文档，未注明各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
```

```
#include<math.h>
```

```
int getmin[2];
void minandAddress(int a,int b)
{
    getmin[0]=a;
    getmin[1]=0;
    if(b<getmin[0])
    {
        getmin[0]=b;
        getmin[1]=1;
    }
}
int maxjoint(int a,int b)
{
    int t,p[2];
    p[0]=a;
    p[1]=b;
    if(p[0]<p[1])
    {
        t=p[0];
        p[0]=p[1];
        p[1]=t;
    }
    do
    {
        t=p[0]%p[1];
        p[0]=p[1];
        p[1]=t;
    }
    while (t!=0);
```

```

    t=p[0];
    return t;
}
void main()
{
    int a[2],b[2],x[2][2]={1,0},{0,-1},value,x0,y0,joint,A,B,flag=1;
    printf("请像这样输入相对来说的,被移动的尺子的最小分度值 a 和被观察的尺子
    的最小分度值 b,之间以逗号隔开'a,b':");
    scanf("%d,%d",a,b);
    A=a[0];
    B=b[0];
    joint=maxjoint(a[0],b[0]);
    minandAddress(a[0],b[0]);
    do
    {
        a[1]=a[0];
        b[1]=b[0];
        if(a[0]%getmin[0]!=0)
            a[0]=a[0]%getmin[0];
        if(b[0]%getmin[0]!=0)
            b[0]=b[0]%getmin[0];
        if(getmin[1]==0)
        {
            x[0][0]=x[0][0]+(b[1]-b[0])/getmin[0]*x[1][0];
            x[0][1]=x[0][1]+(b[1]-b[0])/getmin[0]*x[1][1];
        }
        else
        {
            x[1][0]=x[1][0]+(a[1]-a[0])/getmin[0]*x[0][0];

```

```

        x[1][1]=x[1][1]+(a[1]-a[0])/getmin[0]*x[0][1];
    }
    minandAddress(a[0],b[0]);
    if(getmin[0]==joint)
        break;
}
while(1);
value=x[0][0]*x[1][1]-x[0][1]*x[1][0];
if(getmin[1]==0)
{
    y0=-x[1][0]/value;
    x0=x[1][1]/value;
}
else
{
    y0=x[0][0]/value;
    x0=-x[0][1]/value;
}
if(y0<0)                                /*这里不写≤是为了让当 y0=0 时,flag=1 而
不是 0*/
    flag=-1;
if(abs(y0)<=A/joint/2)
    printf("\nVmax=%d*V 相\nVmin=%d*V 相\n\n",(flag*A*B/joint-
B*y0)/joint,-B*y0/joint);
else
    printf("\n 这个情况应该是不会出现的- -.\n\n");
}

```

/* 这是之前的结尾部分。

```

if(abs(x0)<=B/joint/2)

    printf("\nVmax=%d*V 相\nVmin=%d*V 相\n\n",(flag*A*B/joint-
B*y0)/joint,-B*y0/joint);

    else

    {

        printf("\n 这个情况在不是 a=b 即%d=%d 的时候,应该是不会出现的-
.\n",A,B);

        printf("\nVmax=%d*V 相\nVmin=%d*V 相\n\n",(flag*A*B/joint-
B*y0)/joint,-B*y0/joint);

    }

*/

/* 这是现在的结尾部分。

if(abs(y0)<=A/joint/2)

    printf("\nVmax=%d*V 相\nVmin=%d*V 相\n\n",(flag*A*B/joint-
B*y0)/joint,-B*y0/joint);

    else

        printf("\n 这个情况应该是不会出现的- .\n\n");

*/

```

在以上的小程序和之前的描述中,可以发现我们的算法对于 $ax-by=c$ 中, 当 $c=(a,b)$ 时, 能够百分之百求出距离 $(0,0)$ 最近的最小整数解 (x_0,y_0) 。并且我们发现, 这里最小整数解 (x_0,y_0) 并不等于 “对于 $ax-by=(a,b)$, $x=x_0+t*b/(a,b)$, $y=y_0-t*a/(a,b)$, 均一定有一个 (x_0,y_0) 下的 x_0 、 y_0 值的绝对值同时分别 $\leq b/(a,b)/2$ 、 $a/(a,b)/2$ 。” , 因为后者这个设想本身就有瑕疵: 仅仅当 $a \neq b$ 时, 这个理论才成立, 而不是我们算法的问题。||||| 所以对于我们之前的说法, 两个地方都需要修正为: 对于 $ax-by=c$ (a 、 b 均 >0) ①. “当 $c=(a,b)$ 时, 若 $a \neq b$, 则同一个解 (x_0,y_0) 的 x_0 、 y_0 值的绝对值一定同时分别 $\leq b/(a,b)/2$ 、 $a/(a,b)/2$ 。” 【之前我们的①.没有加条件 $a \neq b$ 】 ②. “当 $c=任意时$,

只能说：同一个解 (x_0, y_0) 的 x_0 、 y_0 值的绝对值不一定同时分别 $\leq b/(a,b)/2$ 、 $a/(a,b)/2$ ；但一定分别存在且只存在一个这样的，即使不在同一个解里的， x_0 、 y_0 。”【之前我们的②.模仿了①.认为是一定存在一个解 (x_0, y_0) 中的 x_0, y_0 同时分别满足~。】

——所以推而广之，该理论可以被更准确的描述为①. “对于一个 $|a| \neq |b|$ 的 $ax+by=\pm(a,b)$ ， $x=x_0+t*b/(a,b)$ ， $y=y_0-t*a/(a,b)$ ，均一定有且只有一个解 (x_0, y_0)

【或者说 (x, y) 】所对应的 x_0 、 y_0 值(或者说 x 、 y 值)的绝对值同时分别 $\leq |b/(a,b)|/2$ 、 $|a/(a,b)|/2$ 。并且我们的任意一个程序能够做到百分之百算出它们。并且连同它们，我们的程序算出的不在规范 $|a| \neq |b|$ 内的解，也均为距离 $(0,0)$ 最近的最小整数解 (x_0, y_0) ”

②. “对于 $ax+by=c$ ， $x=x_0+t*b/(a,b)$ ， $y=y_0-t*a/(a,b)$ ，不一定存在一个解 (x_0, y_0)

【或者说 (x, y) 】所对应的 x_0 、 y_0 值(或者说 x 、 y 值)的绝对值同时分别 $\leq |b/(a,b)|/2$ 、 $|a/(a,b)|/2$ 。但一定分别存在且只存在一个这样的，即使不在同一个解里的， x_0 、 y_0 。这时，“距离 $(0,0)$ 最近的最小整数解 (x_0, y_0) ”有着更为复杂的寻找规则，并且我们在此之前的程序“无法百分之百”(且近乎于0)算出当 c 任意之时距离 $(0,0)$ 最近的解。”

五.二元一次不定方程的两个奇异解之间的，

各种量度标准下的，最小整数解

I.引入

若我们称存在一个解 (x_0, y_0) ，使得其所对应的 x_0 、 y_0 值的绝对值同时分别 $\leq |b/(a,b)|/2$ 、 $|a/(a,b)|/2$ 的方程 $ax+by=c$ 为非奇异方程，并称其所对应的满足条件的那个解 (x_0, y_0) 为非奇异解，且其余所有解既不为奇异解也不为非奇异解。；那么对应的有，若一个方程 $ax+by=c$ ，其任何一个解 (x_0, y_0) 所对应的 x_0 、 y_0 值的绝对值均不同时分别 $\leq |b/(a,b)|/2$ 、 $|a/(a,b)|/2$ ，则称该方程为奇异方程，并称其所一定拥有的两个 (x, y) 中只有一个 x 或 $y \leq$ 对应的 $|b/(a,b)|/2$ 或 $|a/(a,b)|/2$ 的解 (x, y) 为它的两个奇异解。且其余所有解既不为奇异解也不为非奇异解。

这样区分的意义在于，非奇异方程的非奇异解，直接就是“各种意义上的最小整数解”——绝对值之和、绝对值的算术平均值、平方和、平方和的方根(即到原点的距离)等等意义上，均为这些量度标准下的“最小整数解”【不用像奇异解一样，因标准的变化而可能变化，变得新的“最小整数解”不再是自己了，而此时便又要用新标准重新量度出新的标准下的“最小整数解”】；相对的，奇异方程(的奇异解)就没有任何好处了，正如之前括号里所说的，不仅标准的改变会导致最小整数解的重新量度，更可气的是，任何一个标准都会先得到两个“接近最小”的“最小整数解候选人”，而“真正的最小整数解” lies in 它俩以及它俩之间，所以不得不进行至少一次比较。——而这个区间的两个端点就是之前所说的“一定分别存在且只存在一个这样的，即使不在同一个解里的， x_0 、 y_0 ”中的 x_0 、 y_0 所在的两个解 (x, y) 。

一句话概括上面一段话便是：无论何种量度标准，也无论二元一次不定方程的奇异与否，任何一个给定意义下的“最小整数解”，只存在于两个奇异解之间。【若方程非奇异，则视它的那一个非奇异解为“两个相等的奇异解”。】——对于这个定理中的“之间”二字的理解，只能听我娓娓道来了，因为这涉及“等效最小整数解”等等的概念引入了。【不过现在倒是可以先科普科普：“之间”是指：若不失一般性地，设 (x_0, y_0) 为方程 $ax+by=c$ 的一个奇异解，且它所对应的 $|y_0| \leq |a/(a, b)|/2$ ，那么按照 $x=x_0+t*b/(a, b)$ ， $y=y_0-t*a/(a, b)$ 这样的变换规则(若 a 、 b 异号，则记 $(a, b) = -(|a|, |b|)$ 这个东西早就该科普了)，令 t 从 0 开始恒朝着能到达另一个奇异解 (x_1, y_1) 的方向(即向着 $x_0 \rightarrow x_1$ 的方向，即向着能使得 $|x|$ 减少的方向，以至于使得 $|x| \leq |b/(a, b)|/2$ 而让 x 成为 x_1)，要么一直增，要么一直减，至 $x=x_1$ 为止，期间所经过的所有整数 t 所对应的 x 所对应的 (x, y) ，均叫两个奇异解之间的解。而所有意义上的“最小整数解”都至少首先得存在于在这个闭区间上。】

II. 用圆向量求最小整数解

下面就开始介绍一种充分利用圆向量有关知识做出来的，本质性解释了各种意义下的最小整数解由来的，求各种意义下的最小整数解的方法。——这一部分我是先脑子里有了理论雏形，并以之写了程序后，再在这里写的理论，似乎以后也可以这样做哈，条理会非常清晰，但各位就看不到我作为一个真正的人类的一面，而不是神明，在得到真理的路途上的自我纠正过程了；即若这样做的话，不免有同学认为我的得到过程是一帆风顺的；当然，不管前进的道路上有着怎样的错误，只要最终不是逻辑上的错误，那么真正的逻辑迟早会将我们引入真理。——我会直接按照程序的思路呈现出理论的解法步骤：

①.将方程 $ax+by=c$ 强制转化成 $Ax-By=C$ 的形式, 其中, 对 a 、 b 、 c 没有任何限制要求, 但对 A 、 B 、 C , 要求 A 、 $B>0$, $C\geq 0$ 且 $A\geq B$; 另外, 转换将因此导致 (x,y) 的形式变为 $\{(x,y)$ 、 $(x,-y)$ 、 $(-x,y)$ 、 $(-x,-y)$ 、 (y,x) 、 $(y,-x)$ 、 $(-y,x)$ 、 $(-y,-x)\}$ 中的一种。——这种转换是百分之百可行的, 我在这里举几个例子: $3x-5y=-8$ 会变为 $5y-3x=8$; $3x-5y=8$ 会变为 $5(-y)-3(-x)=8$; $3x+5y=-8$ 会变为 $5(-y)-3x=8$; $3x+5y=8$ 会变为 $5y-3(-x)=8$; 以上这些这便是“构造”出来的, 圆向量的标准模型; 这也是为什么只需要研究 $Ax-By=C$ 就可以通晓 $ax+by=c$ 的所有性质的原因。

②.既然已经转化为了圆向量的模型, 那么, 在圆向量中我们说过, 若以 A 为大圆周长, 以 B 为步子的步长, 则步子在所走的 $[A,B]/B$ 那么多步内, 每一步的落脚点互异——于是对应的真实情况(即将这句话翻译成数学语言)便是, 对于每个落脚点所对应的, 比之所走过的路程 By_0 稍大一点的【但大不了一个大于 A 的长度】, 大圆 A (大圆 A 也是个步子)所走过的路程 Ax_0 , 将有 Ax_0-By_0 在不同的、且有效的 (x_0,y_0) 的组合下【其中 $y_0\in\{[0,[A,B]/B]\text{的整数}\}$, $x_0\in\{[1,[A,B]/A]\text{的整数}\}$, 且各自均会取遍自己集合中的元素。】, 将取遍 $i*(A,B)$ 【其中 $i\in\{[0,[A,B]/B]\text{的整数}\}$ 】。

将所有的这些一组组 $Ax_0-By_0=i*(A,B)$ 里面的 x_0 、 y_0 、 $i*(A,B)$ 三个数据为一行【注: 由于 $i*(A,B)=Ax_0-By_0$, 所以程序是以 x_0 、 y_0 、 Ax_0-By_0 为一行】, 从上到下列成一个表格, 其中每一行的行序号= y_0 【从上到下, 行序号和 y_0 均从 1 开始+1】。

③.判断表格中的每一行中的 y_0 是否 $\leq [A,B]/B/2$ 【现在已经修改为判断其是否 $< [A,B]/B/2$ 】, 如果是, 继续判断下一行; 如果否, 则这一行的 $y_0=y_0-[A,B]/B$, 且这一行的 $x_0=x_0-[A,B]/A$ 。——By doing so, 我们把所有行的 (x_0,y_0) 均变为了其所对应的 Ax_0-By_0 所对应的方程的一个奇异解(或非奇异解), 所有 $|y_0|\leq [A,B]/B/2$ 。【注: 我们用的是 y_0 是否 $\leq (<)[A,B]/B/2$, 而不是 x_0 是否 $\leq (<)[A,B]/A/2$ 来限定每一行, 来致使所有行的 $|x_0|\leq [A,B]/A/2$, 原因等下揭晓, 并且这也是这里的精髓所在。】于是我们便生成了一个新的表格, 这个表格等下将非常奏效。——题外话, 为了研究工作的需要, 我生成了两份表格, 一份原表格, 一份新表格, 两者行与行一一对应, 并且在其中标记了解和方程的奇异性: “*****”所在的行, 为奇异方程和奇异解所在的行, “-----”所在的行为非奇异方程和非奇异解所在的行。——这里介绍个快速判断方法判断各个【 $i\in\{[0,[A,B]/B]\text{的整数}\}$ 】的方程 $Ax-By=i*(A,B)$ 的奇异性: 首先, 若 $[A,B]/A$ 为偶数, 则任意 i 所对应的方程均为非奇异方程, 对应的最小整数解均为非奇异解, 即不会出现奇异解; 若是奇数, 则进入下一层判断: 对于 $x_0=[A,B]/A/2+1=[A,B]/A/2+1=([A,B]/A-1)/2+1=([A,B]/A+1)/2$, 和 $y_0=(A*(x_0-1)-A*(x_0-1)\%B)/B+1\sim M$ 【当 $[A,B]/B$ 为偶数时, $M=[A,B]/B/2-1$; 当 $[A,B]/B$ 为奇数时, $M=[A,B]/B/2=[A,B]/B/2=([A,B]/B-1)/2$; 当 $M<(A*(x_0-1)-A*(x_0-1)\%B)/B+1$ 时, 对于各 i 没有奇异解】的组合 (x_0,y_0) , 便是对于【 $i\in\{[0,[A,B]/B]\text{的整数}\}$ 】的所有的

奇异解，其余 i 下的 (x_0, y_0) 全为非奇异解【不过程序里不会通过这种方法来找出它们并标记它们。】；若在之前的 $[A, B]/A$ 为奇数的条件下 $[A, B]/B$ 为偶数，那么会因“ y_0 是否 $\leq [A, B]/B/2$ ”这种判断标准而在表格中间存在一个“伪奇异解”，所以我们得修正“ y_0 是否 $\leq [A, B]/B/2$ ”为“ y_0 是否 $< [A, B]/B/2$ ”，并以之作为新的限制标准：由于若 $y_0 = [A, B]/B/2$ ，减去 $[A, B]/B$ 后仍在范围内，而此时若“下移后的伪奇异解”的 x_0 也在范围内，那么此时“下移后的伪奇异解”其实是个非奇异解；若“下移后的伪奇异解”的 x_0 仍不在范围内，那么此时“下移后的伪奇异解”是个真正的奇异解——这里规定：“伪奇异解”既不是非奇异解也不是奇异解。

④. 由于一个属于 $(-[A, B]/2, [A, B]/2)$ 的 Ax_0 或 By_0 ，加或减了一个 $[A, B]$ 就会导致 $|Ax_0|$ 和 $|By_0| > [A, B]/2$ 而不再属于这个范围，所以一个 $< [A, B]/A/2$ 或一个 $< [A, B]/B/2$ 的 $|x_0|$ 或 $|y_0|$ ，加或减了一个 $[A, B]/A = B/(A, B)$ 或 $[A, B]/B = A/(A, B)$ 就会导致 $|x_0| > [A, B]/A/2 = B/(A, B)/2$ 或 $|y_0| > [A, B]/B/2 = A/(A, B)/2$ 。——这便是为什么最多只有两个奇异解：因为 ax 、 by 或者说 x 、 y ，各自独立地最多只有一次机会进入这个狭窄的范围，玩家一旦按照 $x = x_0 + t \cdot b/(a, b)$ ， $y = y_0 - t \cdot a/(a, b)$ 的规则加或减了一个 $b/(a, b)$ 或 $a/(a, b)$ ，便会出局。【但其实各自独立地最多有两次机会进入这个狭窄的范围，因为 Ax_0 或 By_0 中的(最多)某一个可能属于 $(-[A, B]/2, [A, B]/2)$ 且 $= \pm [A, B]/2$ (不能都同时在边界线/中心线上，因为 $[A, B]/B$ 和 $[A, B]/A$ 不能同时为偶数，否则 $[A, B]$ 会变成 $2[A, B]$ ，且 (A, B) 会变成 $2(A, B)$)】——并且由于 x 、 y 因 t 而联系在了一起，它们各自总会，或朝正向地，或朝负向地，同时变化一个自己所对应的长度 $|b/(a, b)|$ 和长度 $|a/(a, b)|$ ，那么 1. 若当某个解 (x_0, y_0) 所对应的 x_0 在领域里时而 y_0 不在，则当同一个方程所对应的另某个解 (x_1, y_1) 所对应的 y_1 在领域里时【 y 一定有这个机会进入自己的领域，所以若有奇异解的话，就会有两个】， x_1 就将不再在领域里了(除非 x_0 恰好在边境线上)。(因为 y_0 从不在领域变得在领域，必须得朝某个固定方向迈进长度 $|a/(a, b)|$ ，因此，由于 t 的存在， x_0 也会朝某个固定方向迈进长度 $|b/(a, b)|$ ，因而从在领域变得不在领域了(除非 x_0 恰好在边境线上)。) 2. 同理，若当某个解 (x_0, y_0) 所对应的 x_0 在领域里时 y_0 也在这个领域，则当同一个方程所对应的任何一个解 (x, y) 所对应的 x 或 y 不在领域里时，其对应的 y 或 x 就将也不再在领域里了(除非 x_0 或 y_0 恰好在边境线上)。

程序便不再加以解释地充分利用了这个规则，在③.中，既可以先将所有 $|x_0|$ (这儿 x_0 为正，写成 x_0 也可)限定在范围 $(<)$ 内，然后逐个察看 $|y_0|$ 是否在范围 (\leq) 内，若是，则标记为非奇异解所在的行；若不是，则标记它为奇异解所在的行；也可以先将所有 $|y_0|$ (这儿 y_0 为正，写成 y_0 也可)限定在范围 $(<)$ 内，然后逐个察看 $|x_0|$ 是否在范围 (\leq) 内。——然而我们选择了后者：这是为了我们之后的工作做铺垫：即我们现在的工作便需要利用这个之前的选择了——至此我们已经判断了 $Ax - By = C$ 中， $C = i \cdot (A, B)$ 时【 $i \in \{ \in [0, [A, B]/B] \text{ 的整数} \}$ 】的解的奇异情况，那么剩下的便是去寻找当

$C > A$ 时【当然 C 也得被 (A, B) 整除，不然方程无解】的解的奇异情况。——此时就要用到我们之前的“先将所有 y_0 限定在范围内”的设定了：

程序会用 C 去除以 A ，所得的余数为 $C\%A$ 、商为 $(C-C\%A)/A$ 。因为对于任意一个 C ，除以 A 得到的余数 $C\%A$ 恒 $\in \{i*(A, B)\}$ 【 $i \in \{ \in [0, [A, B]/B-1]$ 的整数}】，即必定会等于我们之前所列出的表格中的第 2 行~最后一行中，某一行的 C 值： Ax_0-By_0 。
——接下来，由于 $(1, 0)$ 是方程 $A*1-B*0=A$ 的一个解， (x_0, y_0) 是方程 $Ax_0-By_0=C\%A$ 的一个解，那么 $(C-C\%A)/A*(1, 0)+1*(x_0, y_0)$ ，即 $((C-C\%A)/A+x_0, y_0)$ ，即 $(\text{商}+x_0, y_0)$ ，便是方程 $Ax-By=(C-C\%A)/A*A+1*C\%A$ 的一个解，即是方程 $Ax-By=C$ 的一个解。
——总而言之，方程 $Ax-By=C$ 的一个解是 $(\text{商}+x_0, y_0)$ ，而其中的 (x_0, y_0) 值可通过查找表格中第三列中(即 Ax_0-By_0 或 $i*(A, B)$)的单元格的值为 $C\%A$ 的行，所对应的，第一列和第二列的值： x_0 、 y_0 来得到。

这样的解 $(\text{商}+x_0, y_0)$ 还有什么好处呢：它的 y 值虽然与 C 有关，但是无论 C 是多少， $y=C\%A$ 所对应的 y_0 ，其绝对值恒 $\leq [A, B]/B/2$ ——这是因为我们之前已经先将所有行的 y_0 限定在范围 $(<)$ 内了——所以此解 $(\text{商}+x_0, y_0)$ 至少都会是一个奇异解，并可能会是非奇异解【当 $|\text{商}+x_0| \leq [A, B]/A/2$ 也满足时】。——即不可能既不是奇异解也不是非奇异解。——这便是我们之前选择将所有行的 y_0 限定在范围 $(<)$ 内的原因和好处所在：如果我们不这样做，而是先去将所有行的 x_0 限定在范围 $(<)$ 内，那么这种情况下，如果对于 $C=\text{各个 } i*(A, B)$ 【 $i \in \{ \in [0, [A, B]/B]$ 的整数}】， $Ax-By=C$ 有奇异解的话，由于奇异解的 x_0 已被限定在范围 $(<)$ 内，则奇异解的 $|y_0|$ 会一定 $> [A, B]/B/2$ 【事实上更精确地还有， y_0 会一定 $< -[A, B]/B/2$ ，并且表格中的所有 y_0 均 $\leq [A, B]/B/2$ 。这里我不再解释它的来源，读者自证】，而如果此时又有 $|\text{商}+x_0| > [A, B]/A/2$ ，那么此解 $(\text{商}+x_0, y_0)$ 便既不是奇异解也不是非奇异解了。——即若之前选择限制 x_0 的话，则不一定能保证 $(\text{商}+x_0, y_0)$ 至少是一个奇异解。

⑤.在运用特殊规则保证了 $(\text{商}+x_0, y_0)$ 是方程 $Ax-By=C$ 的一个奇异解【这里把非奇异解也算做奇异解了】后，我又来可以解释一下之前在①.中为什么要限制 $A \geq B$ 了：还记得圆向量中开篇就说了允许 $A < B$ 吗(对于开篇所说“ A 、 B 、 C 、 x 、 y 均可以为负”，我们已经解释过了：因为任意 $ax+by=c$ 都可以化为 $Ax-By=C$)？那么我们要为什么要限制 $A \geq B$ 呢：这么做仅仅是为了能够在列表后，算“绝对值之和最小”标准下的最小整数解的步骤最简单——怎么个最简单法呢：由于 $(\text{商}+x_0, y_0)$ 是个“上奇异解”——即由于 $\text{商} \geq 0$ ，因而 $\text{商}+x_0 \geq x_0$ ，而由于之前将所有行的 y_0 限定在范围 $(<)$ 内的行为，导致了，若方程 $Ax-By=\text{各个 } i*(A, B)$ 有奇异解的话，则奇异解的 $|x_0|$ 会 $> [A, B]/A/2$ 【事实上更精确地还有， x_0 会一定 $> [A, B]/A/2$ 。这里我也不解释它的来源，读者自证】。——利用厚方括号里的内容：奇异解的 x_0 会一定为正且 $> [A, B]/A/2$ ，且又由于其余 $i*(A, B)$ 所对应的方程全为非奇异方程，即其余的 $|x_0| \leq [A, B]/A/2$ ，那么表格中

的所有 x_0 均 $\geq -[A,B]/A/2$ ，这样一来——由于 $\text{商}+x_0 \geq x_0$ ，则 $\text{商}+x_0$ 也一定 $\geq -[A,B]/A/2$ ，并且若 x_0 是奇异解的 x_0 ， $\text{商}+x_0$ 还 $> [A,B]/A/2$ 。——“上奇异解”便是这样的解：每一个方程 $ax+by=c$ 的等效方程 $Ax-By=C$ 中的两个奇异解中的那个 “ $-[A,B]/B/2 \leq y_0 < [A,B]/B/2$ 且 $x_0 \geq -[A,B]/A/2$ 或者 $-[A,B]/A/2 \leq x_0 < [A,B]/A/2$ 且 $y_0 \geq -[A,B]/B/2$ ” 的奇异解所对应的原方程的奇异解；对应的，“下奇异解”便是这两个奇异解中的另一个：“ $-[A,B]/A/2 < x_0 \leq [A,B]/A/2$ 且 $y_0 \leq [A,B]/B/2$ 或者 $-[A,B]/B/2 < y_0 \leq [A,B]/B/2$ 且 $x_0 \leq [A,B]/A/2$ ” 的奇异解所对应的原方程的奇异解。——并且有：既是上奇异解，又是下奇异解的奇异解，为非奇异解。

【插入】：紧接着，推导 I.引入中提及的定理“无论何种量度标准，也无论二元一次不定方程的奇异与否，任何一个给定意义下的“最小整数解”，只存在于两个奇异解之间”的条件也已经成熟：不失一般性地，对于“上奇异解”，它的 $|y_0| \leq [A,B]/B/2$ ，那么 $|y_0 + [A,B]/B| \geq [A,B]/B/2$ 【且有 $y_0 + [A,B]/B \geq [A,B]/B/2$ 】，即这一部分的绝对值在第一次 $+ [A,B]/B$ 时会至少不减，并且因 $y_0 + [A,B]/B \geq [A,B]/B/2 \geq 0$ ，接下来若对这一部分一直 $+ [A,B]/B$ 的话，这一部分的绝对值在之后会一直增大；对于它的另一部分： $x_0 \geq -[A,B]/A/2$ ，那么 $x_0 + [A,B]/A \geq [A,B]/A/2$ ，即这一部分的绝对值在第一次 $+ [A,B]/A$ 时会至少不减，由于这两个部分同加同减，若对上一个部分一直 $+ [A,B]/B$ 的话，这一部分也会一直 $+ [A,B]/A$ ，则同理，那么这一部分的绝对值在之后也会一直增大。——那么这个“绝对值之和”的整体，从一开始就会一直增大（这是因为不可能同时有 $|y_0| = [A,B]/B/2$ 和 $x_0 = -[A,B]/A/2$ ，即不可能两个的第一次均同时不减不增）（但更重要的是，我们要的是这里的：各自的绝对值均不减）。——同理，对于“下奇异解”，若对它的两个部分同减的话，“绝对值之和”也会从一开始就一直增大（且各自的绝对值均不减）。——那么，为了 seek for 各种意义上的最小整数解【若各自的绝对值均不减，那么这些元素的其他映射之和，便不会再是对应的，其他的，通常意义下的，“最小整数解”了】，上奇异解的 x 、 y 只能同减，下奇异解的 x 、 y 只能同加，即“最小整数解”的 (x,y) 只能在上下两个奇异解之间。

介绍完上下奇异解的定义后接着说：由于 $(\text{商}+x_0, y_0)$ 是个“上奇异解”，它只能自减： $(\text{商}+x_0 - t*[A,B]/A, y_0 - t*[A,B]/B)$ 【其中 $t \geq 0$ 】，即“往下走”（ t 从 0 开始递增），才能得到“下奇异解”（因为要想从 $x \geq -[A,B]/A/2$ 变成 $|x'| \leq [A,B]/A/2$ ，只能减去 $t*[A,B]/A$ ，其中 $t \geq 0$ ）。——而这时，限制 $A \geq B$ 的好处就来了：从 $t=1$ 开始，随着 t 的增加， $|\text{商}+x_0 - t*[A,B]/A| + |y_0 - t*[A,B]/B|$ 只增不减！这是因为——当 $t=1$ 时，即当“上奇异解”向下走一步时， $y_0 - [A,B]/B$ 就会跳出 $|y_0| \leq [A,B]/B/2$ 的范围（ y_0 之前是被限制来 $< [A,B]/B/2$ 的），此时以及此后， $y_0 - t*[A,B]/B$ 恒为负，因而当 $t \geq 1$ 时，每当 $t=t+1$ ，这个部分的绝对值就会加上 $[A,B]/B$ ——而对于另一个部分“ $\text{商}+x_0$ ”，若减一个 $[A,B]/A$ 之前之后均为正数，那么这个部分的绝对值就会减去 $[A,B]/A$ ；若减 $[A,B]/A$ 之前为正，之后为负，则这个部分的绝对值会减不了 $[A,B]/A$ 那么多；若减

$[A,B]/A$ 之前和之后均为负，则这个部分的绝对值会反而增加 $[A,B]/A$ ——即 $|\text{商}+x_0-t*[A,B]/A|$ 这个部分，从 $t \geq 0$ 开始， t 每+1，最多减小 $[A,B]/A$ ；而 $|y_0-t*[A,B]/B|$ 这个部分，从 $t \geq 1$ 开始， t 每+1，固定增加 $[A,B]/B$ 。——所以，对于 $|\text{商}+x_0-t*[A,B]/A|+|y_0-t*[A,B]/B|$ 这个整体而言，当 $t \geq 1$ 时， t 每+1，则至少增加 $[A,B]/B-[A,B]/A$ ，又因为之前限制了 $A \geq B$ ，所以 $[A,B]/B-[A,B]/A$ 是一个非负数！所以，当 $t \geq 1$ 时，随着 t 的增加，“绝对值的和”这个关于 t 的函数至少是不减的，而我们要求的是“绝对值的和”的最小值，所以我们只需要比较 $t=0$ 时的函数值和 $t=1$ 时的函数值即可，它俩之中那个较小值，即为所有函数值中的最小值，并且该最小值所对应的整数解为 $Ax-By=C$ 的最小整数解 (x_0, y_0) ，即也为 $ax+by=c$ 的最小整数解 (x, y) 的等效最小整数解 (x_0, y_0) ——此后只需要再用等效最小整数解 $(x_0, y_0) = \{(x, y), (x, -y), (-x, y), (-x, -y), (y, x), (y, -x), (-y, x), (-y, -x)\}$ 中的那某一个的对应法则，得出原最小整数解 (x, y) 。

⑥.在给出了“限制 $A \geq B$ ”对于 $Ax-By=C$ 来说，在求“绝对值之和”意义上的最小整数解，上的便利性后——我们继续在这条分支上走下去，触发此分支的最终剧情：在通晓了这条分支上“ $|\text{商}+x_0-t*[A,B]/A|+|y_0-t*[A,B]/B|$ ”的最小值在哪里取到后，我们再来看看这条分支上的“ $(\text{商}+x_0-t*[A,B]/A)^2+(y_0-t*[A,B]/B)^2$ ”的最小值在哪里取到：此时我们就可以用梦寐以求的“未减初”了—— $[(\text{商}+x_0-(t+1)*[A,B]/A)^2+(y_0-(t+1)*[A,B]/B)^2]-[(\text{商}+x_0-t*[A,B]/A)^2+(y_0-t*[A,B]/B)^2]=-[A,B]/A*(2\text{商}+2x_0-(2t+1)*[A,B]/A)+[A,B]/B*(2y_0-(2t+1)*[A,B]/B)=(2t+1)*([A,B]/A)^2+([A,B]/B)^2-2*[A,B]/A*(\text{商}+x_0)+[A,B]/B*y_0$ ——其中自变量为 t ，整个式子是个一次函数，其系数 $2*([A,B]/A)^2+([A,B]/B)^2$ 为正，单调递增。则当这个差值第一次为非负时，记此时的 t 为 t_0 ，知 t_0+1 的平方和 $\geq t_0$ 时候的平方和，并且因“第一次为非负”，且因 t 的系数为正，则之前均为负，且之后均为非负，由这两点即有： t_0 的平方和 $< t_0-1$ 时候的平方和；且对于 $0 \leq t \leq t_0-1$ ，均有 t 的平方和 $< t-1$ 时候的平方和，并对 $t \geq t_0+1$ 也恒有 t 的平方和 $\geq t-1$ 时候的平方和。——那么，“平方和”意义上的最小整数解，在 $t=t_0$ 时刻取到，而当方括号内为整时 $t_0 = \left\lfloor \frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2} \right\rfloor$ ，此时可以解除方括号，当方括号内不为整时 $t_0 = \left\lfloor \frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2} \right\rfloor + 1$ ，其中的长方括号表示对括号内的运算结果进行取整运算。【关于它的得来：“差值为非负”解出 $t \geq \frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2}$ ，而又要是“第一次为非负”，所以 t_0 为大于等于 t 的最小值的整数，即有以上结果。——在计算机上我会统一两者为同一形式下的公式：

$\left\lfloor \frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2} \right\rfloor + 1 - ! \left(\frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2} - \left\lfloor \frac{[A,B]*(\text{商}+x_0)+[A,B]*y_0}{([A,B]/A)^2+([A,B]/B)^2} - \frac{1}{2} \right\rfloor \right)$ ，若按照数学上的取整方法，那么这个公式已经是通式了；然而计算机的上的取整，当程序

中 $t00 = -\frac{\frac{[A,B]}{A} * (\text{商} + x0) + \frac{[A,B]}{B} * y0}{(\frac{[A,B]}{A})^2 + (\frac{[A,B]}{B})^2} - \frac{1}{2}$ 为负时，本来我们所期望的，(int)t00 在数学上应该等于-1，然而可气的是，计算机却将它认为0。——因此这个公式若要在计算机上施行，还需要一番处理，即令此时的 t 的最小值 t00=0，或者直接就令 t0=0。】

1.程序：二元一次方程的奇异解

此理论(以上 6 点)对应的程序源码如下：(vc++6.0, 205 行, 6.5 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int changehappens,joint;
void list(int *a1,int *a2)
{
    int max=*a1;
    changehappens=0;
    if(*a1<*a2)
    {
        *a1=*a2;
        *a2=max;
        max=*(a1+1);
        *(a1+1)=*(a2+1);
        *(a2+1)=max;
        changehappens=1;
    }
}
void maxjoint(int a1,int a2)
{
    int t;
```



```

do
{
    t=a1%a2;
    a1=a2;
    a2=t;
}
while(t!=0);
joint=a1;
}

void main()
{
    int
a[3][3],**x,i,j,*y=(int*)malloc(sizeof(int)),count,k,x1,x2,real1,real2,real1d,real2d,m
1,m2,n1,n2,cishu=-1,t0;

    float t00;

    printf("请输入被测试的二元一次不定方程 a1x+a2y=a3 的三个系数,中间以逗
号隔开,(不要输多了个逗号或其他符号)例如:'a1,a2,a3:");

    scanf("%d,%d,%d",a[0],a[1],a[2]);    /*举个例子:到这里: 3x-5y=-8 */

    alreadyreinputc:
    for(i=0;i<=2;i++)
    {
        if(a[i][0]<0)
        {
            a[i][1]=-a[i][0];
            a[i][2]=-1;
        }
        else
        {
            a[i][1]=a[i][0];
            a[i][2]=1;

```

```

    }
}
/*到这里:  $3(1*x)+5(-1*y)=8(-1)$  */
list(a[0]+1,a[1]+1);      /*这一行完毕后:  $5(-1*y)+3(1*x)=8(-1)$  */
a[1][2]=-a[1][2];        /*这一行完毕后:  $5(-1*y)-3(-1*x)=8(-1)$  */
if(a[2][2]==-1)
{
    a[2][2]=-a[2][2];
    a[0][2]=-a[0][2];
    a[1][2]=-a[1][2];
}
/*到这里:  $5(1*y)-3(1*x)=8(1)$  */
maxjoint(a[0][1],a[1][1]);
if(a[2][1]%joint!=0)
{

printf("*****\n");

printf("您输入的 a3 无法使该方程有整数解,请重新输入 a3(不要输多了
个逗号或其他符号):");

goto reinputc;
}
cishu++;

if(cishu>=1)              /*如果程序已经至少运行过/经过了一次这个地
方,则用 realloc 否则用 malloc.*/
{
    x=(int***)realloc(x,2*sizeof(int**));
    for(i=0;i<=1;i++)
        x[i]=(int**)realloc(x[i],(a[0][1]/joint+1)*sizeof(int*));
    for(i=0;i<=1;i++)
        for(j=0;j<=a[0][1]/joint;j++)

```

```

x[i][j]=(int*)realloc(x[i][j],3*sizeof(int));
}
else
{
x=(int***)malloc(2*sizeof(int**));
for(i=0;i<=1;i++)
x[i]=(int**)malloc((a[0][1]/joint+1)*sizeof(int*));
for(i=0;i<=1;i++)
for(j=0;j<=a[0][1]/joint;j++)
x[i][j]=(int*)malloc(3*sizeof(int));
}
for(i=1;i<=a[1][1]/joint;i++)    /*初始化*/
for(j=(a[0][1]*(i-1)-a[0][1]*(i-1)%a[1][1])/a[1][1]+1;j<=(a[0][1]*i-
a[0][1]*i%a[1][1])/a[1][1];j++)
{
x[0][j][0]=i;
x[0][j][1]=j;
x[0][j][2]=a[0][1]*i-a[1][1]*j;
}
x[0][0][0]=1;    /*这里相当于 i=1,j=0 的情况,单独列出来.*/
x[0][0][1]=0;
x[0][0][2]=a[0][1];    /*初始化完毕*/
count=0;
for(j=0;j<=a[0][1]/joint;j++)
if(x[0][j][1]*2<a[0][1]/joint)  /*不选择限制 x[0][j][0]而是限制另一个.;保留 x[0][j][1]<a[0][1]/joint/2 的.*/  /*这里要用<而不是<=.*/*
{
x[1][j][0]=x[0][j][0];
x[1][j][1]=x[0][j][1];

```



```

x[1][j][2]=x[0][j][2];

if(x[1][j][0]*2>a[1][1]/joint)/*标记这些不规则/不标准的小朋友,记录它们的下角标j.*/

{

    y=(int*)realloc(y,++count*sizeof(int));

    y[count-1]=j;

}

else

{

    x[1][j][0]=x[0][j][0]-a[1][1]/joint;

    x[1][j][1]=x[0][j][1]-a[0][1]/joint;

    x[1][j][2]=x[0][j][2];

}

printf("\n(%d)x+(%d)y=%d (变换前的方程)\n",a[0][0],a[1][0],a[2][0]); /*
先打印一份我们的对照表*/

if(changehappens)

    printf("%d(%d*y)-%d(%d*x)=%d (变换后的方程)\n\n",a[0][1],a[0][2],a[1][1],a[1][2],a[2][1]);

else

    printf("%d(%d*x)-%d(%d*y)=%d (变换后的方程)\n\n",a[0][1],a[0][2],a[1][1],a[1][2],a[2][1]);

if(count==0)

    for(j=0;j<=a[0][1]/joint;j++)

    {

        printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);

        printf(" ----- ");

        printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);

        if(x[1][j][2]==a[2][1]%a[0][1])

```

```

        k=j;
    }
else
    for(j=0;j<=a[0][1]/joint;j++)
    {
        if(j>=y[0]&& j<=y[count-1])
        {
            printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);
            printf(" ***** ");

printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);
        }
        else
        {
            printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);
            printf(" ----- ");

printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);
        }
        if(x[1][j][2]==a[2][1]%a[0][1])
            k=j;
    }
x1=(a[2][1]-a[2][1]%a[0][1])/a[0][1]+x[1][k][0];
x2=x[1][k][1];
if(abs(x1)*2>a[1][1]/joint)
{
    printf("\n 您所输入的 a3 使得方程的'等效最小整数解'(%d,%d)奇异,所以它现退步为了'两种等效最小整数解,的候选人'.\n\n",x1,x2);

    if(abs(x1)+abs(x2)>abs(x1-a[1][1]/joint)+abs(x2-a[0][1]/joint))
    {

```

```

        m1=x1-a[1][1]/joint;
        m2=x2-a[0][1]/joint;
    }
    else
    {
        m1=x1;
        m2=x2;
    }

    t00=(x1*a[1][1]/joint+x2*a[0][1]/joint)/(pow(a[1][1]/joint,2)+pow(a[0][1]/joint,2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/

    if(t00<0)
        t00=0;

    t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的同时转换类型,以使得分母整个为 float 型.*/

    n1=x1-t0*a[1][1]/joint;
    n2=x2-t0*a[0][1]/joint;
    if(changehappens)
    {
        real1=m2/a[1][2];
        real2=m1/a[0][2];
        real1d=n2/a[1][2];
        real2d=n1/a[0][2];
    }
    else
    {
        real1=m1/a[0][2];
        real2=m2/a[1][2];
        real1d=n1/a[0][2];
        real2d=n2/a[1][2];
    }

```

```

    }

    printf("方程在绝对值的平均数上的'等效最小整数解'为:(%d,%d),真正
的最小整数解为(%d,%d),此时| %d|+| %d|最小.\n",m1,m2,real1,real2,real1,real2);

    printf("方程在离原点(0,0)的距离上的'等效最小整数解'为:(%d,%d),真正
的最小整数解为(%d,%d),此时(%d)^2+(%d)^2 最
小.\n\n",n1,n2,real1d,real2d,real1d,real2d);

    }
    else
    {

        printf("\n 您所输入的 a3 使得(%d,%d)成为了方程的'等效最小整数解',
且它非奇异.\n\n",x1,x2);

        if(changehappens)
        {

            real1=x2/a[1][2];
            real2=x1/a[0][2];

        }
        else
        {

            real1=x1/a[0][2];
            real2=x2/a[1][2];

        }

        printf("并且方程的真正的最小整数解为:(%d,%d).\n 不论从绝对值的平
均数上还是离原点(0,0)的距离上,| %d|+| %d|和(%d)^2+(%d)^2 均最
小.\n\n",real1,real2,real1,real2,real1,real2);

    }

    printf("*****\n");

    printf("请再次输入需要测试的 a3 值(不要输多了个逗号或其他符号:");

    reinputc:

    scanf("%d",a[2]);

```

```
goto alreadyreinputc;
}
```

III. 上一个理论的分身：另一个镜像理论

——让我们更接近真理吧：

(1). 我们的旅程在 II. 中一开始的①. 处就还有一个分支，一个平行世界，一个可能性，尚未涉及：请允许我在这里再介绍介绍，上面的一个理论的分身/同胞兄弟——在①. 和⑤. 中我们均谈到了“限制 $A \geq B$ ”这个约束条件——即我们之前在 II. 中所介绍的一整套理论，是 based on 所创造的 $A \geq B$ 的条件，以之为基础发展而来的，但也同时仅仅是那某一个更大更全的理论的一个分支、一个镜像：

我们在⑤. 中谈到了，1. “圆向量中开篇就说了允许 $A < B$ ”，2. “仅仅是为了算“绝对值之和最小”标准下的最小整数解的步骤最简单”——在第二点中，我用了“仅仅”二字，为的就是和第一点形成照应，为下文我在这里继续介绍另一个原理相似，但优点更显而易见的方法，做铺垫：

i. 它的可行性：若在①. 处就创造一个 $A \leq B$ 的，原方程的等效方程(其他条件均不变)，那么在第②. 处，仍然出于圆向量的模型，大步子 B 在小圆 A (小周长的大圆) 上留下的印记也会遍布小圆上间隔为 (A, B) 的所有刻度，即 $Ax_0 - By_0$ 在不同的、且有效的 (x_0, y_0) 的组合下【其中 $y_0 \in \{ \in [0, [A, B]/B]$ 的整数 $\}$ ， $x_0 \in \{ \in [1, [A, B]/A]$ 的整数 $\}$ ，但 y_0 会取遍它所属的集合中的元素，但 x_0 不会取遍自己集合中的所有元素】，将取遍 $i^*(A, B)$ 【其中 $i \in \{ \in [0, [A, B]/B]$ 的整数 $\}$ 】——即对于任意 C 除以 A 所得的余数，在这个集合 $i^*(A, B)$ 中均找得到对应的值。

ii. 它的优越性：在第③. 处，打表方式跟以前也一样，不过有意思的是，可以见得，表格的行数明显减少了，这归根于我们的设计： $A \leq B$ ，那么行数会有 i 或 y_0 所对应的 $[A, B]/B + 1$ 行，而 B 是较大的一方，则这里 $[A, B]/B + 1$ 是相对于之前的 $[A, B]/B + 1$ 较小的一方。(之前的行数仍然在形式上是 $[A, B]/B + 1$ ，它们的 B 均代表“第二个系数”，但之前的 B 的值为这里的 A 的值)——这就导致了我们想要的：让计算机省省脑子和让我们省省眼睛。——比如，当 $||a| - |b||$ 比较大时，这个分支的做法的效率就非常明显地高了。——另外地，该算法在 $|a| \neq |b|$ 的情况下，不会出现被“*****”所标记的奇异解所在的行。即表格在大多数情况下，都是被“-----”非奇异解所充斥的。它的证明如下：一个未跨越 $[A, B]/2$ 这条中心线的 By_0 刻度(也不在其上，因为若在其上，则 $[A, B]/B$ 为偶，则它是个伪奇异解，但本质上它也是“-----”非奇异解，所以即使 $[A, B]/B$ 为偶，只要 $|a| \neq |b|$ ，则在 i 的范围内均为非奇异解；若超过了，则没

有意义), 和一个 $Ax_0 = By_0 - By_0 \% A$ 的刻度, 均未跨越中心线 $[A, B]/2$, 那么接下来, 由于 $A > By_0 \% A$, 则 $A(x_0+1) = By_0 - By_0 \% A + A > By_0$, 假设它这个刻度超过了中心线 $[A, B]/2$ 【若在其上, 则 $[A, B]/A$ 为偶, 则全非奇异解, 则很好判断是 “-----” ; 若没有在其上也没有超过, 则无意义】, 那么根据对称性, 刻度 $A(x_0+1)$ 与刻度 Ax_0 关于中心线 $[A, B]/2$ 对称, 而刻度 By_0 也将有个 $B(y_0+1)$ 与之关于中心线对称, 而由于 $By_0 > Ax_0$, 那么 $B(y_0+1) < A(x_0+1)$, 进而有 $B(y_0+1) - B(y_0) < A(x_0+1) - Ax_0$, 即有 $B < A$, 然而已有题设 $A \leq B$, 所以此情况不存在, 所以 $|a| \neq |b|$ 时, 没有当 $|c| = i * (A, B)$, $i \in \{ \in [0, [A, B]/B] \text{ 的整数} \}$ 的奇异解。

iii. 它的不足和对应的补救措施: 第④.点和以前一样, 跳过。在第⑤.点的开头, 我们提及了 “ $A \geq B$ ” 能使 “解绝对值之和最小” 标准下的最小整数解的步骤最简单”, 于是这便是 “ $A \leq B$ ” 的缺点所在: 它在 “解绝对值之和最小” 标准下的最小整数解” 上不能像分支 “ $A \geq B$ ” 那样那么直接就能得出, 而是需要先进行 “奇异解与奇异解之间的超时空转换” ——值得注意的是, 这里我们所得到的 (商 $+x_0, y_0$) 也是个上奇异解, 而上奇异解 (x_1, y_1) 到下奇异解 (x_2, y_2) 的转换公式为: $x_2 = (x_1 + [B/(A, B)/2]) \% (B/(A, B)) - [B/(A, B)/2]$, $y_2 = y_1 - A/(A, B) * ((x_1 + [B/(A, B)/2]) \text{ 除以 } (B/(A, B)) \text{ 所得到的商})$ 。【注: 这种方法得到的下奇异解理论上可能为伪奇异解, 即: $-[A, B]/A/2 < x_2 \leq [A, B]/A/2$ 且 $y_2 \leq [A, B]/B/2$ 中可能出现 $x_2 = -[A, B]/A/2$, 即若 $[A, B]/A$ 为偶, 可能出现 $(x_1 + [A, B]/A/2) \% ([A, B]/A) = 0$: 比如输入 13, 22, 297——此时你所得到的下奇异解 $(-11, 20)$ 不是真正的下奇异解 $(11, -7)$, 而是伪奇异解。——解决的办法是: 若 $x_2 = -[A, B]/A/2$, 则 $x_2 = x_2 + [A, B]/A$ 且商 = 商 - 1 即 $y_2 = y_2 + [A, B]/B$ 】当得到转换后的 “下奇异解” 时, 这个解就等价于另一个分支 “ $A \geq B$ ” 中的上奇异解了, 此时便可以利用之前的判定方法, 比较它和它相邻的 (注意, 这里是朝上比较了, 而不是之前的和下面一个相邻的比较), 即比较 $|x_2| + |y_2|$ 与 $|x_2 + B/(A, B)| + |y_2 + A/(A, B)|$, 取其中的较小值为所有中的最小值。——这样看来这一分支的这方面缺陷也不是那么大, 绝对值之和的算法也很简单, 所以总的来说它似乎是没有缺点而只有优点的。【另外, 关于平方和的最小值, 和何时取得, 过程与之前的⑥.完全一样。】

1.程序: 二元一次方程的奇异解—镜像

下面便是分支 “ $A \leq B$ ” 所对应的程序源码: (vc++6.0, 214 行, 6.5 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>
```

其它作品下载链接. (click here) Turn up the sound. (my tunes) chzh_xie@foxmail.com. (contact)

```
int changehappens,joint;
void list(int *a1,int *a2)
{
    int min=*a1;
    changehappens=0;
    if(*a1>*a2)
    {
        *a1=*a2;
        *a2=min;
        min=*(a1+1);
        *(a1+1)=*(a2+1);
        *(a2+1)=min;
        changehappens=1;
    }
}
void maxjoint(int a2,int a1)
{
    int t;
    do
    {
        t=a2%a1;
        a2=a1;
        a1=t;
    }
    while(t!=0);
    joint=a2;
}
void main()
{
```

```

int
a[3][3],**x,i,j,*y=(int*)malloc(sizeof(int)),count,k,x1,x2,real1,real2,real1d,real2d,m
1,m2,n1,n2,cishu=-1,t0,xi1,xi2; /* */

float t00;

printf("请输入被测试的二元一次不定方程 a1x+a2y=a3 的三个系数,中间以逗
号隔开,(不要输多了个逗号或其他符号)例如:'a1,a2,a3:");

scanf("%d,%d,%d",a[0],a[1],a[2]); /*举个例子:到这里: 3x-5y=-8 */

alreadyreinputc:
for(i=0;i<=2;i++)
{
    if(a[i][0]<0)
    {
        a[i][1]=-a[i][0];
        a[i][2]=-1;
    }
    else
    {
        a[i][1]=a[i][0];
        a[i][2]=1;
    }
}

/*到这里: 3(1*x)+5(-1*y)=8(-1) */

list(a[0]+1,a[1]+1); /*这一行完毕后: 3(1*x)+5(-1*y)=8(-1) */

a[1][2]=-a[1][2]; /*这一行完毕后: 3(1*x)-5(1*y)=8(-1) */

if(a[2][2]==-1)
{
    a[2][2]=-a[2][2];
    a[0][2]=-a[0][2];
    a[1][2]=-a[1][2];
}

/*到这里: 3(-1*x)-5(-1*y)=8(1) */

```



```

maxjoint(a[1][1],a[0][1]);
if(a[2][1]%joint!=0)
{

printf("*****\n");

printf("您输入的 a3 无法使该方程有整数解,请重新输入 a3(不要输多了
个逗号或其他符号):");

goto reinputc;
}
cishu++;

if(cishu>=1) /*如果程序已经至少运行过/经过了一次这个地
方,则用 realloc 否则用 malloc.*/
{
x=(int***)realloc(x,2*sizeof(int**));
for(i=0;i<=1;i++)
x[i]=(int**)realloc(x[i],(a[0][1]/joint+1)*sizeof(int*));
for(i=0;i<=1;i++)
for(j=0;j<=a[0][1]/joint;j++)
x[i][j]=(int*)realloc(x[i][j],3*sizeof(int));
}
else
{
x=(int***)malloc(2*sizeof(int**));
for(i=0;i<=1;i++)
x[i]=(int**)malloc((a[0][1]/joint+1)*sizeof(int*));
for(i=0;i<=1;i++)
for(j=0;j<=a[0][1]/joint;j++)
x[i][j]=(int*)malloc(3*sizeof(int));
}

```

```

for(j=0;j<=a[0][1]/joint-1;j++)    /*初始化*/
{
    i=(a[1][1]*j-a[1][1]*j%a[0][1])/a[0][1]+1;
    x[0][j][0]=i;
    x[0][j][1]=j;
    x[0][j][2]=a[0][1]*i-a[1][1]*j;
}

x[0][a[0][1]/joint][0]=a[1][1]/joint; /*这里相当于 j=a[0][1]/joint 的情况,单独
列出来.*/

x[0][a[0][1]/joint][1]=a[0][1]/joint;
x[0][a[0][1]/joint][2]=0;          /*初始化完毕*/

count=0;                          /*__*/
for(j=0;j<=a[0][1]/joint;j++)

    if(x[0][j][1]*2<a[0][1]/joint) /*同样,不选择限制 x[0][j][0]而是限制
另一个.;保留 x[0][j][1]<a[0][1]/joint/2 的.*/ /*这里要用<而不是<=.*/*

    {

        x[1][j][0]=x[0][j][0];
        x[1][j][1]=x[0][j][1];
        x[1][j][2]=x[0][j][2];

/*_↓_*/    if(x[1][j][0]*2>a[1][1]/joint)/*标记这些不规则/不标准的小朋友,记录
它们的下角标 j.——但这里一般不会出现不规则的小朋友,只有当|a|=|b|时.*/

        {

            y=(int*)realloc(y,++count*sizeof(int));
            y[count-1]=j;

        }    /*_↑_*/

    }

else

{

```

```

x[1][j][0]=x[0][j][0]-a[1][1]/joint;
x[1][j][1]=x[0][j][1]-a[0][1]/joint;
x[1][j][2]=x[0][j][2];

}

printf("\n(%d)x+(%d)y=%d (变换前的方程)\n",a[0][0],a[1][0],a[2][0]);  /*
先打印一份我们的对照表*/

if(changehappens)

    printf("%d(%d*y)-%d(%d*x)=%d (变换后的方
程)\n\n",a[0][1],a[0][2],a[1][1],a[1][2],a[2][1]);

else

    printf("%d(%d*x)-%d(%d*y)=%d (变换后的方
程)\n\n",a[0][1],a[0][2],a[1][1],a[1][2],a[2][1]);

if(count==0)                /* __ */
    for(j=0;j<=a[0][1]/joint;j++)
    {

        printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);
        printf(" ----- ");
        printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);
        if(x[1][j][2]==a[2][1]*a[0][1])
            k=j;

    }

else                /* _↓_ */
    for(j=0;j<=a[0][1]/joint;j++)
    {

        if(j>=y[0]&& j<=y[count-1])
        {

            printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);
            printf(" ***** ");

```

```

printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);
    }
    else
    {
        printf("%8d%8d%16d",x[0][j][0],x[0][j][1],x[0][j][2]);
        printf(" ----- ");

printf("%8d%8d%16d\n",x[1][j][0],x[1][j][1],x[1][j][2]);
    }
    if(x[1][j][2]==a[2][1]%a[0][1])
        k=j;
    }
    /*_↑_*/
x1=(a[2][1]-a[2][1]%a[0][1])/a[0][1]+x[1][k][0];
x2=x[1][k][1];
if(abs(x1)*2>a[1][1]/joint)
{
    printf("\n 您所输入的 a3 使得方程的'等效最小整数解'(%d,%d)奇异,所以它现退步为了'两种等效最小整数解,的候选人'.\n\n",x1,x2);

    xi1=(x1+a[1][1]/joint/2)*(a[1][1]/joint)-a[1][1]/joint/2;
    xi2=x2-a[0][1]/joint*((x1+a[1][1]/joint/2)-(x1+a[1][1]/joint/2)*(a[1][1]/joint))/(a[1][1]/joint);
    if(xi1*2==-a[1][1]/joint)    /*这句话等效于 if(xi1==--a[1][1]/joint/2&& a[1][1]/joint%2==0)*/
    {
        xi1=xi1+a[1][1]/joint;
        xi2=xi2+a[0][1]/joint;
    }

    printf("方程的上奇异解为(%d,%d),方程的下奇异解为(%d,%d),等价于反方程的上奇异解(%d,%d).\n\n",x1,x2,xi1,xi2,-xi2,-xi1);

```

```

if(abs(xi1)+abs(xi2)>abs(xi1+a[1][1]/joint)+abs(xi2+a[0][1]/joint))
{
    m1=xi1+a[1][1]/joint;
    m2=xi2+a[0][1]/joint;
}
else
{
    m1=xi1;
    m2=xi2;
}

t00=(x1*a[1][1]/joint+x2*a[0][1]/joint)/(pow(a[1][1]/joint,2)+pow(a[0][1]/joint,2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/

if(t00<0)
    t00=0;

t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的同时转换类型,以使得分母整个为 float 型.*/

n1=x1-t0*a[1][1]/joint;
n2=x2-t0*a[0][1]/joint;
if(changehappens)
{
    real1=m2/a[1][2];
    real2=m1/a[0][2];
    real1d=n2/a[1][2];
    real2d=n1/a[0][2];
}
else
{
    real1=m1/a[0][2];
    real2=m2/a[1][2];

```

```

        real1d=n1/a[0][2];
        real2d=n2/a[1][2];
    }

    printf("方程在绝对值的平均数上的'等效最小整数解'为:(%d,%d),真正
    的最小整数解为(%d,%d),此时| %d|+| %d|最小.\n",m1,m2,real1,real2,real1,real2);

    printf("方程在离原点(0,0)的距离上的'等效最小整数解'为:(%d,%d),真正
    的最小整数解为(%d,%d),此时(%d)^2+(%d)^2 最
    小.\n\n",n1,n2,real1d,real2d,real1d,real2d);
}
else
{

    printf("\n 您所输入的 a3 使得(%d,%d)成为了方程的'等效最小整数解',
    且它非奇异.\n\n",x1,x2);

    printf("方程的上奇异解为(%d,%d),方程的下奇异解为(%d,%d),等价于
    反方程的上奇异解(%d,%d).\n\n",x1,x2,x1,x2,-x2,-x1);

    if(changehappens)
    {
        real1=x2/a[1][2];
        real2=x1/a[0][2];
    }
    else
    {
        real1=x1/a[0][2];
        real2=x2/a[1][2];
    }

    printf("并且方程的真正的最小整数解为:(%d,%d).\n 不论从绝对值的平
    均数上还是离原点(0,0)的距离上,| %d|+| %d|和(%d)^2+(%d)^2 均最
    小.\n\n",real1,real2,real1,real2,real1,real2);
}

```

```

printf("*****\n");

printf("请再次输入需要测试的 a3 值(不要输多了个逗号或其他符号:");

reinputc:

scanf("%d",a[2]);

goto alreadyreinputc;

}

```

六.多个理论和对应的程序的内容及结构的

优化

I.对第一个程序(多元一次不定方程的特解之特殊解法)的改

良

原理：利用超级辗转相除之前所保证的各项系数为正，对超级辗转相除后的各项系数的非系数部分，进行分配系数时，我们采用**第五个程序(全正系数多元一次不定方程的非负整数解)**的解法，来使得各项系数分配比例稍微更合理，使得多元一次不定方程的解的各个坐标值稍微小一点。

1.程序：第 1 个程序的改良版(1+5)

代码如下：(vc++6.0, 303 行, 12 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
```

其它作品下载链接. (click here) Turn up the sound. (my tunes) chzh_xie@foxmail.com. (contact)

```

#include<math.h>

#include<stdlib.h>

int getmin[2],value1=1,sign=1,value2=0,*b,*m,*x;          /*创建一个有两元素的一
维全局数组变量 getmin[2],一个累乘器,一个对换装置,一个累加器.*/

void minandAddress(int *p,int count)                      /*创建一个无返回值函数,获取
一个地址和该地址下的 int 类地址个数*/

{
    int i;
    getmin[0]=p[0];
    getmin[1]=0;
    for(i=1;i<count;i++)
    {
        if(p[i]<getmin[0])                                /*这里确定了最小值将是第
一个最小值,因为如果之后的 p[i]=getmin[0],则 getmin[1]不会改变*/
        {
            getmin[0]=p[i];
            getmin[1]=i;
        }
    }
}

int maxjoint(int *p,int count)                            /*创建函数 maxjoint 调取一串整数,
返回它们的最大公因数数值*/

{
    int j,i,t,*pp=(int*)malloc(count*sizeof(int));        /*p2 这个地址是共用的,为
了保持 p2 内容(即实验者之前的输入顺序)不变,创建一个 p2 的复制品*/

    for(i=0;i<count;i++)                                  /*初始化 p2 的复制品*/
        pp[i]=p[i];

    for(j=count-1;j>0;j--)                                /*刚开始就调换好所有相邻数据
的大小顺序以便之后的辗转相除*/

```



```

{
    for(i=0;i<j;i++)
    {
        if(pp[i]<pp[i+1])
        {
            t=pp[i];
            pp[i]=pp[i+1];
            pp[i+1]=t;
        }
    }
}

for(i=count-2;i>=0;i--)                /*由于之前是从左往右==从大
到小,那么此时得从后往前轮换*/
{
    do
    {
        t=pp[i]%pp[i+1];
        pp[i]=pp[i+1];
        pp[i+1]=t;
    }

    while (t!=0);                        /*跳出循环后 pp[i]就是 pp[i]
和 pp[i+1]的最大公因,而剩下的所有右边的 pp[i+1]会全是 0*/
}

t=pp[0];

free(pp);                                /*不用的早点释放掉*/

pp=NULL;

return t;
}

```

void initialisep4(int *p4,int count) /*本来以为 matrix 函数会改变 p4
的排序状况,因而想设置它来每次调用它来初始化 p4,多心了*/

```
{
    int i;
    for(i=0;i<count;i++)
        p4[i]=i;
}
```

```
void exchange(int *a,int *b)
{
```

```
    int t=*a;
    *a=*b;
    *b=t;
}
```

void matrix(int *j,int min,int max,int **p3) /*输入一组关于列的下角标 j 排
列的地址,地址起始偏移量,地址最末偏移量,用于计算的行列式首地址*/

```
{
    if(min==max)                      /*丰收的标志:该收割此次收获了
*/
    {
        for(int k=0;k<=max;k++)
            value1*=p3[k][j[k]];                      /*累乘器起作用了*/
        value2+=value1*sign;                      /*累加器起作用了*/
        value1=1;                      /*初始化累乘器*/
    }
    else
    {
        for(int i=min;i<=max;i++)
        {
            exchange(j+min,j+i);
        }
    }
}
```

```

        if(i!=min)                                /*不和自己对换的对换称
为有效对换,当有效对换一次时,排列改变奇偶性*/

            sign=-1*sign;

            matrix(j,min+1,max,p3);
            exchange(j+min,j+i);

        if(i!=min)                                /*好东西在这里也得写,不
然就成为坏东西了*/

            sign=-1*sign;

    }

}

int matrixoutput(int *j,int min,int max,int **p3,int **pp3,int k)
{
    int jj,ii;

    value2=0;                                     /*初始化 value2*/

    for(jj=0;jj<=max;jj++)                       /*初始化 pp3 成 p3 的样子*/
        for(ii=0;ii<=max;ii++)
            pp3[ii][jj]=p3[ii][jj];

    /*      for(ii=0;ii<=max;ii++)                打印原始系数矩阵
    {
        for(jj=0;jj<=max;jj++)
            printf("%4d",pp3[ii][jj]);
        printf("\n");
    }*/

    for(ii=0;ii<=max;ii++)                       /*用 b 代换后的系数矩阵*/
        pp3[ii][k]=b[ii];

    /*      printf("\n");                        打印它
    for(ii=0;ii<=max;ii++)

```

```

{
    for(jj=0;jj<=max;jj++)
        printf("%4d",pp3[ii][jj]);
        printf("\n");
    }*/
    matrix(j,min,max,pp3);
    return value2;
}

```

```
void get(int n,int leftovers)
```

```

{
    if(leftovers==0)
    {
    }

```

```
    else if(n==1) /*这个东西逻辑上最好在
```

leftovers==0 的后面,但事实上我们的算法允许它们交换位置.因为可能当 n==2 的时候,进入该函数的 else 里后,发现 x[1]取 j 的最大值时,余数 leftovers-m[1]*j==0,这样会出现进入下一层函数 get(1,0),此时我们希望的是先判断 leftovers 是否等于 0,因为已经找到一个合理解了.而如果此时先判断 n==1,由于 leftovers==0,所以条件 leftovers%m[0]==0 恒为真,使得 x[0]=leftovers/m[0];语句达到和 for(i=0;i<n;i++) x[i]=0;语句相同的初始化 x[0]为 0 的效果,并且其他之后剩下的语句均一样.这样以另一个判断标准判断其为有效非负整数解组,让人感到有点 odd,奇怪.但从逻辑上似乎也说得通,所以在我们的算法上是行得通的.*/

```

{
    if(leftovers%m[0]==0)
    {
        b[x[0]]=leftovers/m[0];
    }
    else
        ; /*这里将不会有此 else 情况出现*/
}

```

```

    }
    else if(m[n-1]>leftovers)

        get(n-1,leftovers);                /*这个要写在 n==1 的后面,
即得先判断是否 n 减来==1.所以上次的只是特殊系数的特殊算法.因为如果出现了当
n==1 的时候 m[0]>leftovers(不知道你输入的各项系数中的最小值 m[0]是多少,而
leftovers 有可能=1,所以有可能当 n==1 的时候却进入了 m[0]>leftovers 的流程),则会
进入下一层函数,出现 m[-1],访问的空间将不在范围内.此时理应进入 n==1 的判断的.
而上一个之所以可行是因为 m[0]恒=1,而 leftovers 不会小于 1(余数难道有小余 1 的?),
所以会直接不满足 m[0]>leftovers,然后进入 n==1 的流程.*/

    else
    {
        b[x[n-1]]=(leftovers-leftovers%m[n-1])/m[n-1];
        get(n-1,leftovers%m[n-1]);
    }
}

/*注:p2 有 count2+1 个空间;有 count2 个数;有 count2-1 个 ai 系数;count=count2-1
即系数个数;count-1 即末位系数的数组序数(下标)*/

int main()
{
    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*pp2,s,count1=0,count2=0,i,sum,j,getmax,*p4,v
    alue2copy,**pp3,flag=1,t;

    printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

    s=getchar();                            /*创建一个 scanf 函数用于实现我
的功能,刚开始就需要值进来,所以不用 dowhile*/

    while(s!='\n')
    {
        if(s<=57&&s>=48)
        {

```

```

count1++;                                /*统计输入了多少个"
假·数字"进来*/

*(p1+count1-1)=s;                        /*将 s 的值放入距离
p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

p1=(char*)realloc(p1,count1+1);          /*将 p1 的空间扩
张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/

}

else if(s=='-')

    flag=-1;                             /*只要输入了负号,不管输
入了多少个,逗号与逗号之间的数均判定为负.*/

    else if(s==',')

    {

        sum=0;
/*****初始化 sum 值,为了
下面的循环*/

        for(i=0;i<=count1-1;i++)

            sum=sum+*(p1+i)-'0')*(int)pow(10,count1-1-i);

        if(flag==-1)

        {

            sum=-sum;

            flag=1;

        }

        count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

        count2++;                         /*统计输入了多少个"
真·数字"进来*/

        *(p2+count2-1)=sum;               /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

```

```

        p2=(int*)realloc(p2,(count2+1)*sizeof(int));  /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

    }

    s=getchar();

}

if(count2-1<2)
{

    printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

    return 0;

}

p3=(int**)malloc((count2-1)*sizeof(int*));          /*除 b 以外有(count2-1)
个系数,所以只需给二维数组的首地址 p3 分配那么多空间,因为只有那么多个非系数部
分(行/一维数组)*/

for(i=0;i<count2-1;i++)
{

    p3[i]=(int*)malloc((count2-1)*sizeof(int));      /*除 b 以外有
(count2-1)个系数,所以只需给一维数组的首地址 p3[i]分配那么多空间,因为只有那么多
个非系数部分里的非系数部分(列/一维数组里的元素)*/

    for(j=0;j<count2-1;j++)

        if(j==i)                                     /*趁此机会把非系数部分的
值给赋了,和下面构成系数部分和非系数部分之内外负号调换*/

            if(p2[i]>0)

                p3[i][j]=1;

            else p3[i][j]=-1;

            else p3[i][j]=0;

        p2[i]=abs(p2[i]);                             /*趁此机会把系数部分取绝
对值,和上面构成系数部分和非系数部分之内外负号调换*/

```

```

    }

    getmax=maxjoint(p2,count2-1);                /*在用 maxjoint 之前得保
证系数是正的,所以 p3 的创造和赋值也得在之前*/

    if(p2[count2-1]%getmax!=0)

    {

        printf("您输入的多元一次不定方程无整数解。 \n");

        for(i=0;i<count2-1;i++)

        {

            free(p3[i]);

            p3[i]=NULL;

        }

        free(p3);

        p3=NULL;

        free(p2);

        p2=NULL;

        free(p1);

        p1=NULL;

        return 0;

    }

    sum=1;                                        /*创建跳出循环的条件:只要这一群
系数中有一个等于上一群(=最开始的)系数的最大公因数,则跳出*/

    pp2=(int*)malloc((count2-1)*sizeof(int));    /*复制一个 p2 的镜像,短
暂保存当前的 p2 下的各地址的值*/

    minandAddress(p2,count2-1);                  /*传计算出来后保存至所开
辟的两个空间里的值给 getmin*/

    do                                           /******进入数学部分
******/

    {

```



```

for(i=0;i<count2-1;i++)          /*好吧,为了之后的简洁,
我们还是全部初始化好了*/

    pp2[i]=p2[i];

for(i=0;i<count2-1;i++)          /*先搞定系数部分,此过程
中不能动最小系数方的非系数部分*/

{

    if(p2[i]%getmin[0]!=0)        /******→→→→
→就是这里,得把原来的 p2[i]!=getmin[0]改为 p2[i]%getmin[0]!=0,即从"不等于"改为
"不整除"为输出方*****/

/*
    pp2[i]=p2[i];                初始化 p2 的复制品,
将此语句放入此 if 下,是因为最小系数方的非系数部分只接受给予方的非系数部分,所以
没必要将 p2 下的内容全拿给 pp2*/

    p2[i]=p2[i]%getmin[0];        /*若为非最小值,
即为输出方,即系数部分输出,而非系数部分不改变;最小值方们系数部分均未改变*/

}

for(j=0;j<count2-1;j++)          /*搞定非系数部分*/

{

    for(i=0;i<count2-1;i++)

    {

        p3[getmin[1]][j]=p3[getmin[1]][j]+(pp2[i]-
p2[i])/getmin[0]*p3[i][j];

    }

}                                  /*当
p2[i]==getmin[0]&&!getmin[1]时,系数部分和非系数部分均不发生改变.此时并集已
经为全.*****→→→→修正为:满足 p2[i]%getmin[0]==0&&!getmin[1]的这些
项的两部分均不改变*****/

    minandAddress(p2,count2-1);

if(getmin[0]==getmax)            /******→→→→
根据修正后的观念,这句话既可以按原来的翻译,也可以翻译为:如果所有项的系数部分

```

均能整除它们的最大公因数,那么根据理论这些项的两部分均不改变,则程序的 mission 已经 accomplished*****/

```

{
    sum=0;

    b=(int*)malloc((count2-1)*sizeof(int));    /*相当于 444333
中的 xi*/

    for(i=0;i<count2-1;i++)
        b[i]=0;

    m=(int*)realloc(m,(count2-1)*sizeof(int));    /*初始化 p2 的
复制品*/

    for(i=0;i<count2-1;i++)
        m[i]=p2[i];

    x=(int*)realloc(x,(count2-1)*sizeof(int));    /*用于记录下角标
的跟随变化*/

    for(i=0;i<count2-1;i++)
        x[i]=i;

    for(j=count2-1-1;j>0;j--)                    /*从左往右==从小到
大*/

    {
        for(i=0;i<j;i++)
        {
            if(m[i]>m[i+1])
            {
                t=m[i];
                m[i]=m[i+1];
                m[i+1]=t;

                t=x[i];                    /*当值交换时,
下角标也跟着交换.*/

                x[i]=x[i+1];

```

```

        x[i+1]=t;
    }
}
}
if(p2[count2-1]<0)
{
    get(count2-1,-p2[count2-1]);
    for(i=0;i<count2-1;i++)
        b[i]=-b[i];
}
else
    get(count2-1,p2[count2-1]);
}
}
while(sum);
p4=(int*)malloc((count2-1)*sizeof(int));
initialisep4(p4,count2-1);
matrix(p4,0,count2-2,p3);
value2copy=value2;

pp3=(int**)malloc((count2-1)*sizeof(int*));           /*创建一个 p3 的 copy,
用于将常数项 b[i]替换掉系数矩阵中的第 j 列矢量*/

for(i=0;i<count2-1;i++)
    pp3[i]=(int*)malloc((count2-1)*sizeof(int));

for(j=0;j<count2-1;j++)                                /*输出*/
{
    printf("x%d=%d\n",j+1,matrixoutput(p4,0,count2-
2,p3,pp3,j)/value2copy);

/*          for(i=0;i<count2-1;i++)                    用于调试中检查程序运行
情况.
```

```
        printf("%d",p4[i]);
        printf("\n%d\n",sign);*/
    }
    for(i=0;i<count2-1;i++)                /*弹出*/
    {
        free(pp3[i]);
        pp3[i]=NULL;
    }
    free(pp3);
    pp3=NULL;
    free(p4);
    p4=NULL;
    free(x);
    x=NULL;
    free(m);
    m=NULL;
    free(b);
    b=NULL;
    free(pp2);
    pp2=NULL;
    for(i=0;i<count2-1;i++)
    {
        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
```

```

    free(p1);
    p1=NULL;
}

```

II.对第二个程序(多元一次不定方程的通解之一般解法)的改良

原理：仍然基于通过第五个程序(全正系数多元一次不定方程的非负整数解)的算法，分配给各步骤下的： $p22[k]x+p2[k+2]y=|y[k+1][i]*p22[k+1]|$ 方程中的 x 、 y 的值更合理和“均匀”一些，使得每一步所得到的类似 $y[k+1][i]$ 的系数稍微小点。

1.程序：第 2 个程序的改良版(2+5)

代码如下：(vc++6.0, 543 行, 20 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int getmin[2],b[2],m[2],z[2];          /*创建一个有两元素的一维全局数组变量 getmin[2]*/

void minandAddress(int p2,int p22,int addressp2)
{
    getmin[0]=p2;
    getmin[1]=addressp2;
    if(p22<getmin[0])
    {
        getmin[0]=p22;
        getmin[1]=addressp2-2;
    }
}

```

```

}

int maxjoint(int p2,int p22)                                /*构造函数 maxjoint 调取一对值,返回
回它们的最大公因数*/
{
    int t,p[2];
    p[0]=p2;
    p[1]=p22;
    if(p[0]<p[1])                                            /*调换数据的大小顺序以便之后的
辗转相除*/
    {
        t=p[0];
        p[0]=p[1];
        p[1]=t;
    }
    do
    {
        t=p[0]%p[1];
        p[0]=p[1];
        p[1]=t;
    }
    while (t!=0);                                           /*跳出循环后 p[0]就是*p2 和*pp2
的最大公因*/
    t=p[0];
    return t;
}

void get(int n,int leftovers)
{
    if(leftovers==0)
    {

```

}

else if(n==1)

/*这个东西逻辑上最好在

leftovers==0 的后面,但事实上我们的算法允许它们交换位置.因为可能当 n==2 的时候,进入该函数的 else 里后,发现 x[1]取 j 的最大值时,余数 leftovers-m[1]*j==0,这样会出现进入下一层函数 get(1,0),此时我们希望的是先判断 leftovers 是否等于 0,因为已经找到一个合理解了.而如果此时先判断 n==1,由于 leftovers==0,所以条件 leftovers%m[0]==0 恒为真,使得 x[0]=leftovers/m[0];语句达到和 for(i=0;i<n;i++) x[i]=0;语句相同的初始化 x[0]为 0 的效果,并且其他之后剩下的语句均一样.这样以另一个判断标准判断其为有效非负整数解组,让人感到有点 odd,奇怪.但从逻辑上似乎也说得通,所以在我们的算法上是行得通的.*/

{

if(leftovers%m[0]==0)

{

b[z[0]]=leftovers/m[0];

}

else

;

/*这里将不会有此 else 情况

出现*/

}

else if(m[n-1]>leftovers)

get(n-1,leftovers);

/*这个要写在 n==1 的后面,

即得先判断是否 n 减来==1.所以上次的只是特殊系数的特殊算法.因为如果出现了当 n==1 的时候 m[0]>leftovers(不知道你输入的各项系数中的最小值 m[0]是多少,而 leftovers 有可能=1,所以有可能当 n==1 的时候却进入了 m[0]>leftovers 的流程),则会进入下一层函数,出现 m[-1],访问的空间将不在范围内.此时理应进入 n==1 的判断的.而上一个之所以可行是因为 m[0]恒=1,而 leftovers 不会小于 1(余数难道有小余 1 的?),所以会直接不满足 m[0]>leftovers,然后进入 n==1 的流程.*/

else

{

b[z[n-1]]=(leftovers-leftovers%m[n-1])/m[n-1];

get(n-1,leftovers%m[n-1]);

```

    }
}
/*****/

int main()
{
    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*p22,**p33,pp2[2],s,count1=0,count2=0,i,sum,k
    ,value2copy,**x,**y,*p222,*pp22,flag=1,t;

    printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
    a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

    s=getchar();                                /*创建一个 scanf 函数用于实现我
    的功能,刚开始就需要值进来,所以不用 dowhile*/

    while(s!='\n')
    {
        if(s<=57&&s>=48)
        {
            count1++;                            /*统计输入了多少个"
            假·数字"进来*/

            *(p1+count1-1)=s;                    /*将 s 的值放入距离
            p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

            p1=(char*)realloc(p1,count1+1);      /*将 p1 的空间扩
            张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/

        }
        else if(s=='-')
            flag=-1;
        else if(s==',')
        {

```



```

sum=0;
/*****初始化 sum 值,为了
下面的循环*/

for(i=0;i<=count1-1;i++)
    sum=sum+*(p1+i)-'0'*(int)pow(10,count1-1-i);
if(flag==-1)
{
    sum=-sum;
    flag=1;
}

count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

count2++; /*统计输入了多少个"
真·数字"进来*/

*(p2+count2-1)=sum; /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一个
空间,不过不影响我们的继续,所以循环外不用改小一格它*/

}

s=getchar();

}

if(count2-1<2)
{

printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

return 0;

}

```

```

p3=(int**)malloc((count2-1)*sizeof(int*));          /*从 a0 到 an-1 的非系数
部分:x0 到 xn-1*/

for(i=0;i<count2-1;i++)

    p3[i]=(int*)malloc(2*sizeof(int));              /*现在每个子地址只开
辟两个空间了,每对新创造的最小公倍数和右系数只构成一个二元一次不定式,两个系数,
两个非系数部分.这里只给 x 所在的非系数部分分配空间.*/

    pp22=(int*)malloc((count2-1)*sizeof(int));      /*创造 pp22 来复制 p2,
保留它们的原始值,以后有用。并且要在它们的负号消失之前及时得到最原始的信息.不
能丢失此负号信息.*/

    for(i=0;i<=count2-1-1;i++)

        pp22[i]=p2[i];

/*****/

    for(i=0;i<=1;i++)                                /*先初始化 a0 和 a1 的系数部分
和非系数部分,这里可以沿用之前的语法,之后就不行了*/

    {

        for(k=0;k<=1;k++)

            if(k==i)

                if(p2[i]>0)

                    p3[i][k]=1;

                else p3[i][k]=-1;

            else p3[i][k]=0;

        p2[i]=abs(p2[i]);

    }

/*****/

    if(count2-1>2)

    {

        p22=(int*)malloc((count2-1-2)*sizeof(int)); /*从 b0 到 bn-3 的系
数部分,这是模仿 p2 创造的,但是 p2 还包含有 b 以及一个空空间.*/

```

p33=(int**)malloc((count2-1-2)*sizeof(int*)); /*从 b0 到 bn-3 的
非系数部分:y0 到 yn-3*/

for(i=0;i<count2-1-2;i++)

p33[i]=(int*)malloc(2*sizeof(int)); /*其子地址也只开
辟两个空间*/

}

else /*城乡结合部:综合最初(k=n-3)与最
末(k=-1)二阶式子的语句,具体而微的程序雏形.*/

{

if(p2[count2-1]%maxjoint(p2[0],p2[1])!=0)

{

printf("您输入的多元一次不定方程无整数解。 \n");

free(pp22);

pp22=NULL;

for(i=0;i<count2-1;i++)

{

free(p3[i]);

p3[i]=NULL;

}

free(p3);

p3=NULL;

free(p2);

p2=NULL;

free(p1);

p1=NULL;

return 0;

}

x=(int**)malloc((count2-1)*sizeof(int*));

sum=1;

```

minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0];                                /*保留值*/
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {
        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));

if(pp22[0]*pp22[1]>0)                                /*这里才是真迹:如果系数

```

异号,则规定它们的最大公因数为负值.* /

```

        flag=maxjoint(p2[1],p2[0]);
    else
        flag=-maxjoint(p2[1],p2[0]);
    x[1][count2-2]=pp22[0]/flag;
    x[0][count2-2]=-pp22[1]/flag;
    for(i=0;i<2;i++)                                /*新的替代品__始端*/
        b[i]=0;
    for(i=0;i<2;i++)
        m[i]=p2[i];
    for(i=0;i<2;i++)
        z[i]=i;
    if(m[0]>m[1])
    {
        t=m[0];
        m[0]=m[1];
        m[1]=t;
        t=z[0];
        /*当值交换时,下角标也跟着交换.*/
        z[0]=z[1];
        z[1]=t;
    }
    if(p2[count2-1]<0)
    {
        get(2,-p2[count2-1]);
        for(i=0;i<2;i++)
            b[i]=-b[i];
    }
    else
        get(2,p2[count2-1]);

```

```

for(i=count2-3;i>=0;i--)
{
    x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
    x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;
}
/*新的替代品__末端*/
printf("x%d=",1);
for(i=count2-2;i>=1;i--)
    printf("%dt%d+",x[0][i],i);
printf("%d",x[0][0]);
printf("\n");
for(k=1;k<=count2-2;k++)
{
    printf("x%d=",k+1);
    for(i=count2-1-k;i>=1;i--)
        printf("%dt%d+",x[k][i],i);
    printf("%d",x[k][0]);
    printf("\n");
}
free(x[0]);
x[0]=NULL;
free(x[1]);
x[1]=NULL;
free(x);
x=NULL;
free(pp22);
pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);

```

```

        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}

/*****

    p22[0]=maxjoint(p2[0],p2[1]);           /*没办法,任何设计都有局限,
这里多出了一些周期之外的东西,加上上面的,有两部分周期之外的,这里除了这第一句
话,相当于下面循环中 k=0 的时候的.*/

    p33[0][0]=1;
    p33[0][1]=0;
    p3[2][0]=0;
    if(p2[2]>0)
        p3[2][1]=1;
    else p3[2][1]=-1;
    p2[2]=abs(p2[2]);

*****/

    for(k=1;k<=count2-1-3;k++)
    {
        p22[k]=maxjoint(p2[k+1],p22[k-1]);

        p33[k][0]=1;           /*p22[k]由于是上一对系数的
最大公因数,恒>0,所以直接 p33[k][0]=1,剩下的=0,以及不用取系数的绝对值了*/

        p33[k][1]=0;
        p3[k+2][0]=0;

```

```

        if(p2[k+2]>0)
            p3[k+2][1]=1;
        else p3[k+2][1]=-1;
        p2[k+2]=abs(p2[k+2]);
    }

/*****/

    if(p2[count2-1]%maxjoint(p2[count2-2],p22[count2-4])!=0)
    {

        printf("您输入的多元一次不定方程无整数解。 \n");
        for(i=0;i<count2-1-2;i++)
        {
            free(p33[i]);
            p33[i]=NULL;
        }
        free(p33);
        p33=NULL;
        free(p22);
        p22=NULL;
        free(pp22);
        pp22=NULL;
        for(i=0;i<count2-1;i++)
        {
            free(p3[i]);
            p3[i]=NULL;
        }
        free(p3);
        p3=NULL;
        free(p2);
        p2=NULL;
    }

```



```

    free(p1);
    p1=NULL;
    return 0;
}
x=(int**)malloc((count2-1)*sizeof(int*));
y=(int**)malloc((count2-1-2)*sizeof(int*));
p222=(int*)malloc((count2-1-2)*sizeof(int));      /*创造 p222 来复制 p22,
保留它们的原始值,以后有用,注:由于 p22 全是正的,所以 p222 也全是正的.*/
for(i=0;i<=count2-1-3;i++)
    p222[i]=p22[i];
/*****/      /*相当于 k=n-
3*/
sum=1;
minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
do
{
    pp2[0]=p22[count2-1-3];      /*保留值*/
    pp2[1]=p2[count2-1-3+2];
    if(p22[count2-1-3]%getmin[0]!=0)
        p22[count2-1-3]=p22[count2-1-3]%getmin[0];
    if(p2[count2-1-3+2]%getmin[0]!=0)
        p2[count2-1-3+2]=p2[count2-1-3+2]%getmin[0];
    if(getmin[1]==count2-1-3)
    {
        p33[count2-1-3][0]=p33[count2-1-3][0]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][0];
        p33[count2-1-3][1]=p33[count2-1-3][1]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][1];
    }
    else

```

```

{
    p3[count2-1-3+2][0]=p3[count2-1-3+2][0]+(p2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][0];

    p3[count2-1-3+2][1]=p3[count2-1-3+2][1]+(p2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][1];

}

minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);

if(getmin[0]==maxjoint(p2[count2-1-3+2],p22[count2-1-3]))

    sum=0;

}

while(sum);

value2copy=p33[count2-1-3][0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*p3[count2-1-3+2][0];

x[count2-1-3+2]=(int*)malloc(2*sizeof(int));

y[count2-1-3]=(int*)malloc(2*sizeof(int));

if(pp22[count2-1-3+2]>0)                                /*这里本来是判断
p222[count2-1-3]*pp22[count2-1-3+2]>0,但由于 p222[count2-1-3]本身就已经人为
规定>0了,所以能让程序跑快点就让它跑快点嘛.*/

    flag=maxjoint(p2[count2-1-3+2],p22[count2-1-3]);

else

    flag=-maxjoint(p2[count2-1-3+2],p22[count2-1-3]);

x[count2-1-3+2][1]=p222[count2-1-3]/flag;

y[count2-1-3][1]=-pp22[count2-1-3+2]/flag;

for(i=0;i<2;i++)
    /*新的替代品__始端*/

    b[i]=0;

m[0]=p22[count2-1-3];                                /*细微的差别*/

m[1]=p2[count2-1-3+2];

for(i=0;i<2;i++)

    z[i]=i;

if(m[0]>m[1])

```

```

{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;
}
if(p2[count2-1]<0)
{
    get(2,-p2[count2-1]);
    for(i=0;i<2;i++)
        b[i]=-b[i];
}
else
    get(2,p2[count2-1]);
{
    x[count2-1-3+2][0]=(p33[count2-1-3][0]*b[1]-b[0]*p3[count2-1-
3+2][0])/value2copy;
    y[count2-1-3][0]=(b[0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*b[1])/value2copy;
}
/*新的替代品__末端*/
/*****/
for(k=count2-1-3-1;k>=0;k--)
{
    sum=1;
    minandAddress(p2[k+2],p22[k],k+2);
    do

```

```

{
    pp2[0]=p22[k];                                /*保留值*/
    pp2[1]=p2[k+2];
    if(p22[k]%getmin[0]!=0)
        p22[k]=p22[k]%getmin[0];
    if(p2[k+2]%getmin[0]!=0)
        p2[k+2]=p2[k+2]%getmin[0];
    if(getmin[1]==k)
    {
        p33[k][0]=p33[k][0]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][0];
        p33[k][1]=p33[k][1]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][1];
    }
    else
    {
        p3[k+2][0]=p3[k+2][0]+(pp2[0]-
p22[k])/getmin[0]*p33[k][0];
        p3[k+2][1]=p3[k+2][1]+(pp2[0]-
p22[k])/getmin[0]*p33[k][1];
    }
    minandAddress(p2[k+2],p22[k],k+2);
    if(getmin[0]==maxjoint(p2[k+2],p22[k]))
        sum=0;
}
while(sum);
value2copy=p33[k][0]*p3[k+2][1]-p33[k][1]*p3[k+2][0];
x[k+2]=(int*)malloc((count2-1-k-1)*sizeof(int));
y[k]=(int*)malloc((count2-1-k-1)*sizeof(int));
if(pp22[k+2]>0)
    flag=maxjoint(p2[k+2],p22[k]);

```

```

else
    flag=-maxjoint(p2[k+2],p22[k]);
x[k+2][count2-1-k-2]=p222[k]/flag;
y[k][count2-1-k-2]=-pp22[k+2]/flag;

m[0]=p22[k];                                /*新的替代品__始端;细微的
差别;只有它们能写在外面;那么现在开始放在里面的许多与 i 无关的 i 得改为 t*/
m[1]=p2[k+2];
for(i=0;i<2;i++)
    z[i]=i;
if(m[0]>m[1])
{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;
}
for(i=count2-1-k-3;i>=0;i--)
{
    for(t=0;t<2;t++)                        /*结构改变了,(从这开
始都)它跑里面来了,每次使用都需要初始化 b.*/
        b[t]=0;
    if(y[k+1][i]<0)                          /*这里本该写作
y[k+1][i]*p222[k+1]<0 的*/
    {
        get(2,-p222[k+1]*y[k+1][i]);
        for(t=0;t<2;t++)

```

```

        b[t]=-b[t];
    }
    else
        get(2,p222[k+1]*y[k+1][i]);
    x[k+2][i]=(p33[k][0]*b[1]-b[0]*p3[k+2][0])/value2copy;
    y[k][i]=(b[0]*p3[k+2][1]-p33[k][1]*b[1])/value2copy;
}
/*新的替代品__末端*/
}

/*****/
/*相当于 k=-1*/

sum=1;
minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0];
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {
        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
}

```

```

    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));

if(pp22[0]*pp22[1]>0)                                /*这里才是真迹:如果系数异号,
则规定它们的最大公因数为负值.*/

    flag=maxjoint(p2[1],p2[0]);
else
    flag=-maxjoint(p2[1],p2[0]);
x[1][count2-2]=pp22[0]/flag;
x[0][count2-2]=-pp22[1]/flag;
for(i=0;i<2;i++)
    /*新的替代品__始端;同样,只有它们能写在外面;*/
    m[i]=p2[i];
for(i=0;i<2;i++)
    z[i]=i;
if(m[0]>m[1])
{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;

```

```

    }
    for(i=count2-3;i>=0;i--)
    {
        for(t=0;t<2;t++)
            b[t]=0;
        if(y[0][i]<0)
        {
            get(2,-p222[0]*y[0][i]);
            for(t=0;t<2;t++)
                b[t]=-b[t];
        }
        else
            get(2,p222[0]*y[0][i]);
        x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
        x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;
    }
    /*新的替代品__末端*/
/*****/

    printf("x%d=",1);
    for(i=count2-2;i>=1;i--)
        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
    for(k=1;k<=count2-2;k++)
    {
        printf("x%d=",k+1);
        for(i=count2-1-k;i>=1;i--)
            printf("%dt%d+",x[k][i],i);
        printf("%d",x[k][0]);
        printf("\n");
    }

```



```

    }
/*****/

    free(x[0]);
    x[0]=NULL;
    free(x[1]);
    x[1]=NULL;
    for(k=0;k<=count2-1-3-1;k++)
    {
        free(y[k]);
        y[k]=NULL;
        free(x[k+2]);
        x[k+2]=NULL;
    }
    free(y[count2-1-3]);
    y[count2-1-3]=NULL;
    free(x[count2-1-3+2]);
    x[count2-1-3+2]=NULL;
    free(p222);
    p222=NULL;
    free(y);
    y=NULL;
    free(x);
    x=NULL;
    for(i=0;i<count2-1-2;i++)
    {
        free(p33[i]);
        p33[i]=NULL;
    }
    free(p33);

```

```

p33=NULL;
free(p22);
p22=NULL;
free(pp22);
pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;
}
free(p3);
p3=NULL;
free(p2);
p2=NULL;
free(p1);
p1=NULL;
}

```

III.对第之前的第 II.个程序(或者说是第 10 个程序)的再次

改良

绝对值之和意义上的多元一次不定方程的最小整数解以及遍历其他整数解的最小系数解。

原理：加入第 8 个程序的算法，对 $p22[k]x + p2[k+2]y = |y[k+1][i] * p222[k+1]|$ ，求得这些每一个二元一次方程下的绝对值之和意义上的最小整数解，并让它们作为 b 值参与下一次运算。在这里有一个新的突发情况和应对它的技巧：在第 8 个程序中，我们的一来就会得到一个等效方程的上奇异解，即这个解一来就在下奇异解的上方，故那个“将任意一个 $A \leq B$ 的方程中 $x1 \geq -B/(A,B)/2$ 的解转换到下奇异解”(不仅仅局限于奇异解到奇异解的转换，也可以是非奇异非非奇异的解到奇异解的转换)的公式

【 $x2=(x1+[B/(A,B)/2])\%(B/(A,B))-[B/(A,B)/2]$, $y2=y1-A/(A,B)*((x1+[B/(A,B)/2])$
除以 $(B/(A,B))$ 所得到的商)。且：若 $x2=-[A,B]/A/2$, 则 $x2=x2+[A,B]/A$ 且
 $y2=y2+[A,B]/B$ 】是可行的。

然而在第 10 个程序的改良过程中, 若这么做会出现问题: 我们的程序解出的是个“比较小的随机解”, 它在等效方程中所对应的等效解, 不一定满足条件 $x1 \geq -B/(A,B)/2$ ——但其实这种情况的解决方案相对来说还要更简单一些: 它的公式直接就是【 $-x2=(-x1+[B/(A,B)/2])\%(B/(A,B))-[B/(A,B)/2]$, $y2=y1+A/(A,B)*((-x1+[B/(A,B)/2])$ 除以 $(B/(A,B))$ 所得到的商)】——这个新公式对于任意 $x1 \leq -B/(A,B)/2$ 是普适的, 【其实这个新公式关于旧公式对称, 它在更大的区间 $x1 \leq B/(A,B)/2$ 上都是普适的。】不像之前的公式, 对于 $x1 = -B/(A,B)/2$ 的情况得单列出来。——对于这里的技巧和来源, 你可以如下思考: 因为 $(x1+[B/(A,B)/2])\%(B/(A,B))$ 在 $[0, B/(A,B))$ 的区间上, 所以 $(x1+[B/(A,B)/2])\%(B/(A,B))-[B/(A,B)/2]$ 在 $[-B/(A,B)/2, B/(A,B)/2)$ 这个左闭右开的区间上——然而下奇异解的定义是: “ $-[A,B]/A/2 < x0 \leq [A,B]/A/2$ 且 $y0 \leq [A,B]/B/2$ ”, 它的 $x0$ 是左开右闭的, 所以当解得 $x1 = -B/(A,B)/2$ 时, 得把它变到右区间端点, 因而会多一个步骤。——而由于 $-x2=(-x1+[B/(A,B)/2])\%(B/(A,B))-[B/(A,B)/2]$ 得到的解 $-x2$ 在区间 $[-B/(A,B)/2, B/(A,B)/2)$ 上, 所以 $x2$ 在 $(-B/(A,B)/2, B/(A,B)/2]$ 这个左开右闭的区间上, 直接就是下奇异解了。——我之前就是这样设计公式的(看出来我用了一些“向量”的想法没? : 相对于自设 0 刻度的大小)。

1.程序：第 10 个程序的改良版(2+5+8a)

以下便是它的代码, 它带来的好处不多说, 在我眼里, 是最能最快地求得绝对值相对最小的多元一次不定方程的最有效的算法了: (vc++6.0, 894 行, 33 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int getmin[2],b[2],m[2],z[2],changehappens;          /*创建一个有两元素的一维
全局数组变量 getmin[2]*

void list(int *a1,int *a2)
{
    int min=*a1;
```

```

changehappens=0;
if(*a1>*a2)
{
    *a1=*a2;
    *a2=min;
    min=*(a1+1);
    *(a1+1)=*(a2+1);
    *(a2+1)=min;
    changehappens=1;
}
}
void minandAddress(int p2,int p22,int addressp2)
{
    getmin[0]=p2;
    getmin[1]=addressp2;
    if(p22<getmin[0])
    {
        getmin[0]=p22;
        getmin[1]=addressp2-2;
    }
}
int maxjoint(int p2,int p22)                                /*创建函数 maxjoint 调取一对值,返回它们的最大公因数*/
{
    int t,p[2];
    p[0]=p2;
    p[1]=p22;
    if(p[0]<p[1])                                            /*调换数据的大小顺序以便之后的辗转相除*/

```

```

{
    t=p[0];
    p[0]=p[1];
    p[1]=t;
}
do
{
    t=p[0]%p[1];
    p[0]=p[1];
    p[1]=t;
}
while (t!=0);
/*跳出循环后 p[0]就是*p2 和*pp2
的最大公因*/
t=p[0];
return t;
}
void get(int n,int leftovers)
{
    if(leftovers==0)
    {

```

else if(n==1) /*这个东西逻辑上最好在
 leftovers==0 的后面,但事实上我们的算法允许它们交换位置.因为可能当 n==2 的时候,
 进入该函数的 else 里后,发现 x[1]取 j 的最大值时,余数 leftovers-m[1]*j==0,这样会出
 现进入下一层函数 get(1,0),此时我们希望的是先判断 leftovers 是否等于 0,因为已经找
 到一个合理解了.而如果此时先判断 n==1,由于 leftovers==0,所以条件
 leftovers%m[0]==0 恒为真,使得 x[0]=leftovers/m[0];语句达到和 for(i=0;i<n;i++)
 x[i]=0;语句相同的初始化 x[0]为 0 的效果,并且其他之后剩下的语句均一样.这样以另
 一个判断标准判断其为有效非负整数解组,让人感到有点 odd,奇怪.但从逻辑上似乎也
 说得通,所以在我们的算法上是行得通的.*/
 }

```

{
    if(leftovers%m[0]==0)
    {
        b[z[0]]=leftovers/m[0];
    }
    else
        ; /*这里将不会有此 else 情况
出现:不会除不尽.*/

}
else if(m[n-1]>leftovers)

    get(n-1,leftovers); /*这个要写在 n==1 的后面,
即得先判断是否 n 减来==1.所以上次的只是特殊系数的特殊算法.因为如果出现了当
n==1 的时候 m[0]>leftovers(不知道你输入的各项系数中的最小值 m[0]是多少,而
leftovers 有可能=1,所以有可能当 n==1 的时候却进入了 m[0]>leftovers 的流程),则会
进入下一层函数,出现 m[-1],访问的空间将不在范围内.此时理应进入 n==1 的判断的.
而上一个之所以可行是因为 m[0]恒=1,而 leftovers 不会小于 1(余数难道有小余 1 的?),
所以会直接不满足 m[0]>leftovers,然后进入 n==1 的流程.*/

else
{
    b[z[n-1]]=(leftovers-leftovers%m[n-1])/m[n-1];
    get(n-1,leftovers%m[n-1]);
}
}
/*****/

int main()
{
    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*p22,**p33,pp2[2],s,count1=0,count2=0,i,sum,k
    ,value2copy,**x,**y,*p222,*pp22,flag=1,t,a[3][2],c[2][2],x1,x2,xi1,xi2;

```

```

printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

s=getchar();                                /*创建一个 scanf 函数用于实现我
的功能,刚开始就需要值进来,所以不用 dowhile*/

while(s!='\n')
{
    if(s<=57&&s>=48)
    {
        count1++;                            /*统计输入了多少个"
假·数字"进来*/

        *(p1+count1-1)=s;                    /*将 s 的值放入距离
p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

        p1=(char*)realloc(p1,count1+1);      /*将 p1 的空间扩
张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/
    }
    else if(s=='-')
        flag=-1;
    else if(s==',')
    {
        sum=0;
        /*****初始化 sum 值,为了
下面的循环*/

        for(i=0;i<=count1-1;i++)
            sum=sum+*(p1+i)-'0')*(int)pow(10,count1-1-i);

        if(flag==-1)
        {
            sum=-sum;
            flag=1;
        }
    }
}

```

```

        count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

        count2++;                                /*统计输入了多少个"
真·数字"进来*/

        *(p2+count2-1)=sum;                        /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

        p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

    }
    s=getchar();
}
if(count2-1<2)
{

    printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");

    return 0;

}

    p3=(int**)malloc((count2-1)*sizeof(int*));        /*从 a0 到 an-1 的非系数
部分:x0 到 xn-1*/

    for(i=0;i<count2-1;i++)

        p3[i]=(int*)malloc(2*sizeof(int));            /*现在每个子地址只开
辟两个空间了,每对新创造的最小公倍数和右系数只构成一个二元一次不定式,两个系数,
两个非系数部分.这里只给 x 所在的非系数部分分配空间.*/

        pp22=(int*)malloc((count2-1)*sizeof(int));    /*创造 pp22 来复制 p2,
保留它们的原始值,以后有用。并且要在它们的负号消失之前及时得到最原始的信息.不
能丢失此负号信息.*/

        for(i=0;i<=count2-1-1;i++)

            pp22[i]=p2[i];

```



```

/*****/

    for(i=0;i<=1;i++)                /*先初始化 a0 和 a1 的系数部分
和非系数部分,这里可以沿用之前的语法,之后就不行了*/

    {

        for(k=0;k<=1;k++)

            if(k==i)

                if(p2[i]>0)

                    p3[i][k]=1;

                else p3[i][k]=-1;

            else p3[i][k]=0;

        p2[i]=abs(p2[i]);

    }

/*****/

    if(count2-1>2)

    {

        p22=(int*)malloc((count2-1-2)*sizeof(int));    /*从 b0 到 bn-3 的系
数部分,这是模仿 p2 创造的,但是 p2 还包含有 b 以及一个空空间.*/

        p33=(int**)malloc((count2-1-2)*sizeof(int**));    /*从 b0 到 bn-3 的
非系数部分:y0 到 yn-3*/

        for(i=0;i<count2-1-2;i++)

            p33[i]=(int*)malloc(2*sizeof(int));    /*其子地址也只开
辟两个空间*/

    }

    else                                /*城乡结合部:综合最初(k=n-3)与最
末(k=-1)二阶式子的语句,具体而微的程序雏形.*/

    {

        if(p2[count2-1]%maxjoint(p2[0],p2[1])!=0)

        {

            printf("您输入的多元一次不定方程无整数解。 \n");

```

```

    free(pp22);
    pp22=NULL;
    for(i=0;i<count2-1;i++)
    {
        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}
x=(int**)malloc((count2-1)*sizeof(int*));
sum=1;
minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0];                /*保留值*/
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {
        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];

```

```

        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));
if(pp22[0]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-pp22[0];
    a[0][1]=-1;
}
else
{
    a[0][0]=pp22[0];
    a[0][1]=1;
}
if(pp22[1]<0)
{
    a[1][0]=-pp22[1];
    a[1][1]=-1;
}

```

```

    }
    else
    {
        a[1][0]=pp22[1];
        a[1][1]=1;
    }
    list(a[0],a[1]);
    a[1][1]=-a[1][1];
    for(i=0;i<2;i++)
        c[i][1]=a[i][1];                                /*insert 第二部分,部分尾:
保留原非系数部分的符号.*/

    if(pp22[0]*pp22[1]>0)                                /*这里才是真迹:如果系数
异号,则规定它们的最大公因数为负值.*/

        flag=maxjoint(p2[1],p2[0]);
    else
        flag=-maxjoint(p2[1],p2[0]);
    x[1][count2-2]=pp22[0]/flag;
    x[0][count2-2]=-pp22[1]/flag;
    for(i=0;i<2;i++)                                    /*新的替代品__始端*/
        b[i]=0;
    for(i=0;i<2;i++)
        m[i]=p2[i];
    for(i=0;i<2;i++)
        z[i]=i;
    if(m[0]>m[1])
    {
        t=m[0];
        m[0]=m[1];
        m[1]=t;
    }

```

```

t=z[0];
/*当值交换时,下角标也跟着交换.*/

z[0]=z[1];
z[1]=t;
}
if(p2[count2-1]<0)
{
    get(2,-p2[count2-1]);
    for(i=0;i<2;i++)
        b[i]=-b[i];
}
else
    get(2,p2[count2-1]);
for(i=count2-3;i>=0;i--)
{
    x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
    x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;
    if(p2[count2-1]<0)                /*insert 第二部分,部分
首.*/

    {
        a[2][0]=-p2[count2-1];
        a[2][1]=-1;
    }
    else
    {
        a[2][0]=p2[count2-1];
        a[2][1]=1;
    }
    if(a[2][1]==-1)

```

```

{
    a[2][1]=-a[2][1];
    a[0][1]=-a[0][1];
    a[1][1]=-a[1][1];
}

/*    if(changehappens)

        printf("%d(%d*y)-%d(%d*x)=%d (变换后的方程)\n\n",a[0][0],a[0][1],a[1][0],a[1][1],a[2][0]);

    else

        printf("%d(%d*x)-%d(%d*y)=%d (变换后的方程)\n\n",a[0][0],a[0][1],a[1][0],a[1][1],a[2][0]); */

    if(changehappens)
    {
        x1=x[1][i]*a[0][1];
        x2=x[0][i]*a[1][1];
    }
    else
    {
        x1=x[0][i]*a[0][1];
        x2=x[1][i]*a[1][1];
    }

    if(x1*2<=-a[1][0]/getmin[0])                                /*新的判断标准
和对应的执行标准*/

    {

        xi1=a[1][0]/getmin[0]/2-(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]);    /*本身是-xi1=(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-a[1][0]/getmin[0]/2*/

        xi2=x2+a[0][0]/getmin[0]*((-x1+a[1][0]/getmin[0]/2)-
(-x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);

    }

```

```

else
{
    xi1=(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-
a[1][0]/getmin[0]/2;
    xi2=x2-a[0][0]/getmin[0]*((x1+a[1][0]/getmin[0]/2)-
(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
    if(xi1*2==a[1][0]/getmin[0])
    {
        xi1=xi1+a[1][0]/getmin[0];
        xi2=xi2+a[0][0]/getmin[0];
    }
}

/*      printf("方程的等效随机解为(%d,%d),方程的等效下奇异解为(%d,%d),等价于
反方程的上奇异解(%d,%d).\n\n",x1,x2,xi1,xi2,-xi2,-xi1); */

    if(abs(xi1)+abs(xi2)>abs(xi1+a[1][0]/getmin[0])+abs(xi2+a[0][0]/getmin[0]
))
    {
        xi1=xi1+a[1][0]/getmin[0];
        xi2=xi2+a[0][0]/getmin[0];
    }
    if(changehappens)
    {
        x[0][i]=xi2/a[1][1];
        x[1][i]=xi1/a[0][1];
    }
    else
    {
        x[0][i]=xi1/a[0][1];
        x[1][i]=xi2/a[1][1];
    }
}

```

```

for(t=0;t<2;t++)
    a[t][1]=c[t][1];                                /*insert 第二部分,部
分尾:还原 a[0][1]和 a[1][1].*/

    }                                                /*新的替代品__末端*/

printf("x%d=",1);
for(i=count2-2;i>=1;i--)
    printf("%dt%d+",x[0][i],i);
printf("%d",x[0][0]);
printf("\n");
for(k=1;k<=count2-2;k++)
{
    printf("x%d=",k+1);
    for(i=count2-1-k;i>=1;i--)
        printf("%dt%d+",x[k][i],i);
    printf("%d",x[k][0]);
    printf("\n");
}
free(x[0]);
x[0]=NULL;
free(x[1]);
x[1]=NULL;
free(x);
x=NULL;
free(pp22);
pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;

```



```

    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}

/*****/

    p22[0]=maxjoint(p2[0],p2[1]);           /*没办法,任何设计都有局限,
这里多出了一些周期之外的东西,加上上面的,有两部分周期之外的,这里除了这第一句
话,相当于下面循环中 k=0 的时候的.*/

    p33[0][0]=1;
    p33[0][1]=0;
    p3[2][0]=0;
    if(p2[2]>0)
        p3[2][1]=1;
    else p3[2][1]=-1;
    p2[2]=abs(p2[2]);

/*****/

    for(k=1;k<=count2-1-3;k++)
    {
        p22[k]=maxjoint(p2[k+1],p22[k-1]);

        p33[k][0]=1;           /*p22[k]由于是上一对系数的
最大公因数,恒>0,所以直接 p33[k][0]=1,剩下的=0,以及不用取系数的绝对值了*/

        p33[k][1]=0;
        p3[k+2][0]=0;
        if(p2[k+2]>0)

```

```

        p3[k+2][1]=1;
    else p3[k+2][1]=-1;
    p2[k+2]=abs(p2[k+2]);
}

/*****/

if(p2[count2-1]%maxjoint(p2[count2-2],p22[count2-4])!=0)
{
    printf("您输入的多元一次不定方程无整数解。 \n");
    for(i=0;i<count2-1-2;i++)
    {
        free(p33[i]);
        p33[i]=NULL;
    }
    free(p33);
    p33=NULL;
    free(p22);
    p22=NULL;
    free(pp22);
    pp22=NULL;
    for(i=0;i<count2-1;i++)
    {
        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
}

```

```

    p1=NULL;
    return 0;
}
x=(int**)malloc((count2-1)*sizeof(int*));
y=(int**)malloc((count2-1-2)*sizeof(int*));

p22=(int*)malloc((count2-1-2)*sizeof(int));      /*创造 p22 来复制 p22,
保留它们的原始值,以后有用,注:由于 p22 全是正的,所以 p22 也全是正的.*/

for(i=0;i<=count2-1-3;i++)
    p22[i]=p2[i];

/*****/      /*相当于 k=n-
3*/

sum=1;
minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
do
{
    pp2[0]=p22[count2-1-3];      /*保留值*/
    pp2[1]=p2[count2-1-3+2];
    if(p22[count2-1-3]%getmin[0]!=0)
        p22[count2-1-3]=p22[count2-1-3]%getmin[0];
    if(p2[count2-1-3+2]%getmin[0]!=0)
        p2[count2-1-3+2]=p2[count2-1-3+2]%getmin[0];
    if(getmin[1]==count2-1-3)
    {
        p33[count2-1-3][0]=p33[count2-1-3][0]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][0];
        p33[count2-1-3][1]=p33[count2-1-3][1]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][1];
    }
    else
    {

```

```

        p3[count2-1-3+2][0]=p3[count2-1-3+2][0]+(p2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][0];

        p3[count2-1-3+2][1]=p3[count2-1-3+2][1]+(p2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][1];

    }

    minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);

    if(getmin[0]==maxjoint(p2[count2-1-3+2],p22[count2-1-3]))

        sum=0;

    }

    while(sum);

    value2copy=p33[count2-1-3][0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*p3[count2-1-3+2][0];

    x[count2-1-3+2]=(int*)malloc(2*sizeof(int));

    y[count2-1-3]=(int*)malloc(2*sizeof(int));

    if(p222[count2-1-3]<0)                                /*insert 第二部分首*/

    {

        a[0][0]=-p222[count2-1-3];

        a[0][1]=-1;

    }

    else

    {

        a[0][0]=p222[count2-1-3];

        a[0][1]=1;

    }

    if(pp22[count2-1-3+2]<0)

    {

        a[1][0]=-pp22[count2-1-3+2];

        a[1][1]=-1;

    }

    else

```

```

{
    a[1][0]=pp22[count2-1-3+2];
    a[1][1]=1;
}
list(a[0],a[1]);
a[1][1]=-a[1][1];
for(i=0;i<2;i++)

    c[i][1]=a[i][1];                                /*insert 第二部分,部分尾:保
留原非系数部分的符号.*/

    if(pp22[count2-1-3+2]>0)                            /*这里本来是判断
p222[count2-1-3]*pp22[count2-1-3+2]>0,但由于 p222[count2-1-3]本身就已经人为
规定>0 了,所以能让程序跑快点就让它跑快点嘛.*/

        flag=maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    else

        flag=-maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    x[count2-1-3+2][1]=p222[count2-1-3]/flag;
    y[count2-1-3][1]=-pp22[count2-1-3+2]/flag;
    for(i=0;i<2;i++)
        /*新的替代品__始端*/

        b[i]=0;

    m[0]=p22[count2-1-3];                                /*细微的差别*/
    m[1]=p2[count2-1-3+2];
    for(i=0;i<2;i++)

        z[i]=i;
    if(m[0]>m[1])

    {

        t=m[0];
        m[0]=m[1];
        m[1]=t;

```

```

t=z[0];
/*当值交换时,下角标也跟着交换.*/

z[0]=z[1];
z[1]=t;
}
if(p2[count2-1]<0)
{
    get(2,-p2[count2-1]);
    for(i=0;i<2;i++)
        b[i]=-b[i];
}
else
    get(2,p2[count2-1]);
{
    x[count2-1-3+2][0]=(p33[count2-1-3][0]*b[1]-b[0]*p3[count2-1-
3+2][0])/value2copy;
    y[count2-1-3][0]=(b[0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*b[1])/value2copy;
    if(p2[count2-1]<0)                /*insert 第二部分,部分
首.*/
    {
        a[2][0]=-p2[count2-1];
        a[2][1]=-1;
    }
    else
    {
        a[2][0]=p2[count2-1];
        a[2][1]=1;
    }
    if(a[2][1]==-1)

```

```

{
    a[2][1]=-a[2][1];
    a[0][1]=-a[0][1];
    a[1][1]=-a[1][1];
}
if(changehappens)
{
    x1=x[count2-1-3+2][0]*a[0][1];
    x2=y[count2-1-3][0]*a[1][1];
}
else
{
    x1=y[count2-1-3][0]*a[0][1];
    x2=x[count2-1-3+2][0]*a[1][1];
}
if(x1*2<=-a[1][0]/getmin[0])                /*新的判断标准和对应
的执行标准*/
{
    xi1=a[1][0]/getmin[0]/2-(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]); /*本身是-xi1=(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-a[1][0]/getmin[0]/2*/
    xi2=x2+a[0][0]/getmin[0]*((-x1+a[1][0]/getmin[0]/2)-(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
}
else
{
    xi1=(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-
a[1][0]/getmin[0]/2;
    xi2=x2-a[0][0]/getmin[0]*((x1+a[1][0]/getmin[0]/2)-
(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
    if(xi1*2==a[1][0]/getmin[0])

```

```

    {
        xi1=xi1+a[1][0]/getmin[0];
        xi2=xi2+a[0][0]/getmin[0];
    }
}

if(abs(xi1)+abs(xi2)>abs(xi1+a[1][0]/getmin[0])+abs(xi2+a[0][0]/getmin[0]
))
{
    xi1=xi1+a[1][0]/getmin[0];
    xi2=xi2+a[0][0]/getmin[0];
}
if(changehappens)
{
    y[count2-1-3][0]=xi2/a[1][1];
    x[count2-1-3+2][0]=xi1/a[0][1];
}
else
{
    y[count2-1-3][0]=xi1/a[0][1];
    x[count2-1-3+2][0]=xi2/a[1][1];
}
for(t=0;t<2;t++)
    a[t][1]=c[t][1];
还原 a[0][1]和 a[1][1].*/
}

/*新的替代品__末端*/

/*****/

for(k=count2-1-3-1;k>=0;k--)
{

```

/*insert 第二部分,部分尾:


```

sum=1;
minandAddress(p2[k+2],p22[k],k+2);
do
{
    pp2[0]=p22[k];                /*保留值*/
    pp2[1]=p2[k+2];
    if(p22[k]%getmin[0]!=0)
        p22[k]=p22[k]%getmin[0];
    if(p2[k+2]%getmin[0]!=0)
        p2[k+2]=p2[k+2]%getmin[0];
    if(getmin[1]==k)
    {
        p33[k][0]=p33[k][0]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][0];
        p33[k][1]=p33[k][1]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][1];
    }
    else
    {
        p3[k+2][0]=p3[k+2][0]+(pp2[0]-
p22[k])/getmin[0]*p33[k][0];
        p3[k+2][1]=p3[k+2][1]+(pp2[0]-
p22[k])/getmin[0]*p33[k][1];
    }
    minandAddress(p2[k+2],p22[k],k+2);
    if(getmin[0]==maxjoint(p2[k+2],p22[k]))
        sum=0;
}
while(sum);
value2copy=p33[k][0]*p3[k+2][1]-p33[k][1]*p3[k+2][0];
x[k+2]=(int*)malloc((count2-1-k-1)*sizeof(int));

```

```

y[k]=(int*)malloc((count2-1-k-1)*sizeof(int));
if(p222[k]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-p222[k];
    a[0][1]=-1;
}
else
{
    a[0][0]=p222[k];
    a[0][1]=1;
}
if(pp22[k+2]<0)
{
    a[1][0]=-pp22[k+2];
    a[1][1]=-1;
}
else
{
    a[1][0]=pp22[k+2];
    a[1][1]=1;
}
list(a[0],a[1]);
a[1][1]=-a[1][1];
for(i=0;i<2;i++)
    c[i][1]=a[i][1];                        /*insert 第二部分,部分尾:
保留原非系数部分的符号.*/
if(pp22[k+2]>0)
    flag=maxjoint(p2[k+2],p22[k]);
else

```

```

        flag=-maxjoint(p2[k+2],p22[k]);
        x[k+2][count2-1-k-2]=p22[k]/flag;
        y[k][count2-1-k-2]=-pp22[k+2]/flag;

        m[0]=p22[k];                                /*新的替代品__始端;细微的
        差别;只有它们能写在外面;那么现在开始放在里面的许多与 i 无关的 i 得改为 t*/
        m[1]=p2[k+2];
        for(i=0;i<2;i++)
            z[i]=i;
        if(m[0]>m[1])
        {
            t=m[0];
            m[0]=m[1];
            m[1]=t;
            t=z[0];
            /*当值交换时,下角标也跟着交换.*/
            z[0]=z[1];
            z[1]=t;
        }
        for(i=count2-1-k-3;i>=0;i--)
        {
            for(t=0;t<2;t++)                            /*结构改变了,(从这开
            始都)它跑里面来了,每次使用都需要初始化 b.*/
                b[t]=0;

            if(y[k+1][i]<0)                                /*这里本该写作
            y[k+1][i]*p22[k+1]<0 的*/
            {
                get(2,-p22[k+1]*y[k+1][i]);
                for(t=0;t<2;t++)
                    b[t]=-b[t];
            }
        }
    
```

```

    }
    else
        get(2,p222[k+1]*y[k+1][i]);
    x[k+2][i]=(p33[k][0]*b[1]-b[0]*p3[k+2][0])/value2copy;
    y[k][i]=(b[0]*p3[k+2][1]-p33[k][1]*b[1])/value2copy;
    if(y[k+1][i]<0)                                /*insert 第二部分,部分
首.*/
    {
        a[2][0]=-y[k+1][i]*p222[k+1];
        a[2][1]=-1;
    }
    else
    {
        a[2][0]=y[k+1][i]*p222[k+1];
        a[2][1]=1;
    }
    if(a[2][1]==-1)
    {
        a[2][1]=-a[2][1];
        a[0][1]=-a[0][1];
        a[1][1]=-a[1][1];
    }
    if(changehappens)
    {
        x1=x[k+2][i]*a[0][1];
        x2=y[k][i]*a[1][1];
    }
    else
    {

```

```

x1=y[k][i]*a[0][1];
x2=x[k+2][i]*a[1][1];
}
if(x1*2<=-a[1][0]/p222[k+1])          /*新的判断标准和
对应的执行标准*/
{
    xi1=a[1][0]/p222[k+1]/2-(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]);  /*本身是-xi1=(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1])-a[1][0]/p222[k+1]/2*/
    xi2=x2+a[0][0]/p222[k+1]*((-
x1+a[1][0]/p222[k+1]/2)-(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]))/(a[1][0]/p222[k+1]);
}
else
{
    xi1=(x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1])-
a[1][0]/p222[k+1]/2;
    xi2=x2-
a[0][0]/p222[k+1]*((x1+a[1][0]/p222[k+1]/2)-
(x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]))/(a[1][0]/p222[k+1]);
    if(xi1*2==a[1][0]/p222[k+1])
    {
        xi1=xi1+a[1][0]/p222[k+1];
        xi2=xi2+a[0][0]/p222[k+1];
    }
}

if(abs(xi1)+abs(xi2)>abs(xi1+a[1][0]/p222[k+1])+abs(xi2+a[0][0]/p222[k+
1]))
{
    xi1=xi1+a[1][0]/p222[k+1];
    xi2=xi2+a[0][0]/p222[k+1];
}

```

```

if(changehappens)
{
    y[k][i]=xi2/a[1][1];
    x[k+2][i]=xi1/a[0][1];
}
else
{
    y[k][i]=xi1/a[0][1];
    x[k+2][i]=xi2/a[1][1];
}
for(t=0;t<2;t++)

    a[t][1]=c[t][1];                                /*insert 第二部分,部
分尾:还原 a[0][1]和 a[1][1].*/

}                                                    /*新的替代品__末端*/

}

/*****/                                              /*相当于 k=-1*/

sum=1;
minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0];                                    /*保留值*/
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {

```

```

        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));
if(pp22[0]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-pp22[0];
    a[0][1]=-1;
}
else
{
    a[0][0]=pp22[0];
    a[0][1]=1;
}
if(pp22[1]<0)
{
    a[1][0]=-pp22[1];

```

```

        a[1][1]=-1;
    }
    else
    {
        a[1][0]=pp22[1];
        a[1][1]=1;
    }
    list(a[0],a[1]);
    a[1][1]=-a[1][1];
    for(i=0;i<2;i++)

        c[i][1]=a[i][1];                                /*insert 第二部分,部分尾:保
留原非系数部分的符号.*/

    if(pp22[0]*pp22[1]>0)                                /*这里才是真迹:如果系数异号,
    则规定它们的最大公因数为负值.*/

        flag=maxjoint(p2[1],p2[0]);
    else

        flag=-maxjoint(p2[1],p2[0]);
    x[1][count2-2]=pp22[0]/flag;
    x[0][count2-2]=-pp22[1]/flag;
    for(i=0;i<2;i++)
        /*新的替代品__始端;同样,只有它们能写在外面;*/

        m[i]=p2[i];
    for(i=0;i<2;i++)
        z[i]=i;
    if(m[0]>m[1])
    {
        t=m[0];
        m[0]=m[1];
        m[1]=t;

```



```

t=z[0];
    /*当值交换时,下角标也跟着交换.*/

z[0]=z[1];
z[1]=t;
}
for(i=count2-3;i>=0;i--)
{
    for(t=0;t<2;t++)
        b[t]=0;
    if(y[0][i]<0)
    {
        get(2,-p222[0]*y[0][i]);
        for(t=0;t<2;t++)
            b[t]=-b[t];
    }
    else
        get(2,p222[0]*y[0][i]);
    x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
    x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;
    if(y[0][i]<0)                                /*insert 第二部分,部分首.*/
    {
        a[2][0]=-y[0][i]*p222[0];
        a[2][1]=-1;
    }
    else
    {
        a[2][0]=y[0][i]*p222[0];
        a[2][1]=1;
    }
}

```

```

if(a[2][1]==-1)
{
    a[2][1]=-a[2][1];
    a[0][1]=-a[0][1];
    a[1][1]=-a[1][1];
}
if(changehappens)
{
    x1=x[1][i]*a[0][1];
    x2=x[0][i]*a[1][1];
}
else
{
    x1=x[0][i]*a[0][1];
    x2=x[1][i]*a[1][1];
}
if(x1*2<=-a[1][0]/p222[0])                                /*新的判断标准和对应
的执行标准*/
{
    xi1=a[1][0]/p222[0]/2-(-
x1+a[1][0]/p222[0]/2)%a[1][0]/p222[0]); /*本身是-xi1=(-
x1+a[1][0]/p222[0]/2)%a[1][0]/p222[0]-a[1][0]/p222[0]/2*/
    xi2=x2+a[0][0]/p222[0]*((-x1+a[1][0]/p222[0]/2)-(-
x1+a[1][0]/p222[0]/2)%a[1][0]/p222[0]))/(a[1][0]/p222[0]);
}
else
{
    xi1=(x1+a[1][0]/p222[0]/2)%a[1][0]/p222[0]-
a[1][0]/p222[0]/2;
    xi2=x2-a[0][0]/p222[0]*((x1+a[1][0]/p222[0]/2)-
(x1+a[1][0]/p222[0]/2)%a[1][0]/p222[0]))/(a[1][0]/p222[0]);
}

```

```

        if(xi1*2==-a[1][0]/p222[0])
        {
            xi1=xi1+a[1][0]/p222[0];
            xi2=xi2+a[0][0]/p222[0];
        }
    }

    if(abs(xi1)+abs(xi2)>abs(xi1+a[1][0]/p222[0])+abs(xi2+a[0][0]/p222[0]))
    {
        xi1=xi1+a[1][0]/p222[0];
        xi2=xi2+a[0][0]/p222[0];
    }
    if(changehappens)
    {
        x[0][i]=xi2/a[1][1];
        x[1][i]=xi1/a[0][1];
    }
    else
    {
        x[0][i]=xi1/a[0][1];
        x[1][i]=xi2/a[1][1];
    }
    for(t=0;t<2;t++)

        a[t][1]=c[t][1];                                     /*insert 第二部分,部分尾:
还原 a[0][1]和 a[1][1].*/

    }                                                         /*新的替代品__末端*/

/*****/

    printf("x%d=",1);
    for(i=count2-2;i>=1;i--)

```

```

        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
    for(k=1;k<=count2-2;k++)
    {
        printf("x%d=",k+1);
        for(i=count2-1-k;i>=1;i--)
            printf("%dt%d+",x[k][i],i);
        printf("%d",x[k][0]);
        printf("\n");
    }
/*****/

    free(x[0]);
    x[0]=NULL;
    free(x[1]);
    x[1]=NULL;
    for(k=0;k<=count2-1-3-1;k++)
    {
        free(y[k]);
        y[k]=NULL;
        free(x[k+2]);
        x[k+2]=NULL;
    }
    free(y[count2-1-3]);
    y[count2-1-3]=NULL;
    free(x[count2-1-3+2]);
    x[count2-1-3+2]=NULL;
    free(p222);
    p222=NULL;

```

```

free(y);
y=NULL;
free(x);
x=NULL;
for(i=0;i<count2-1-2;i++)
{
    free(p33[i]);
    p33[i]=NULL;
}
free(p33);
p33=NULL;
free(p22);
p22=NULL;
free(pp22);
pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;
}
free(p3);
p3=NULL;
free(p2);
p2=NULL;
free(p1);
p1=NULL;
}

```

/*pp22 是 p2 的复制品,保留了它的符号,是纯 a 类系数;然后 p2 符号一点点地全为正了,且用 pp2 记录了 p2 和 p2、p2 和 pp2 的最大公因数的 b 类系数;p222 是 p22 的复制品,

即是一系列 b 类系数和最大公因数; p_2 与 p_2 、 p_2 与 p_{22} 之后会进行超级辗转相除, 丢失信息, pp_{22} 和 p_{222} 的价值便体现出来了: 原始的三元一次不定方程们的 3 个系数.*/

/* $p_{222}[k]x + pp_{22}[k+2]y = y[k+1][i] * p_{222}[k+1]$ */

/* $p_{22}[k]x + p_2[k+2]y = |y[k+1][i] * p_{222}[k+1]|$ */

/* $m[z[1]]$ 是超级辗转相除后的 p_2 与 p_2 、 p_2 与 p_{22} 中的较大的系数, $b[z[1]]$ 是它的非系数部分, 即超级辗转后的全正系数方程的解.*/

IV. 对第之前的第 II. 个程序(或者说是第 10 个程序)的另一种

方向上的改良

平方和意义上的多元一次不定方程的最小整数解以及遍历其他整数解的最小系数解。

原理: 并无大异。只需要将 III. 中的程序里的公式修改成求平方和之最小值的就行了——但是若我们想沿用上一个程序, 上一个程序只有下奇异解, 而第 8 个程序中用的是上奇异解, 所以这里我们得稍微修改下平方和的公式来对应这次: “未减初”—— $[(x_1 + (t+1) * [A, B]/A)^2 + (y_1 + (t+1) * [A, B]/B)^2] - [(x_1 + t * [A, B]/A)^2 + (y_1 - t * [A, B]/B)^2] = (2t+1) * ([A, B]/A)^2 + ([A, B]/B)^2 + 2\{[A, B]/A * x_1 + [A, B]/B * y_1\} \geq 0$ 。

$$\text{—— } t \geq -\frac{\frac{[A, B]}{A} * (\text{商} + x_0) + \frac{[A, B]}{B} * y_0}{(\frac{[A, B]}{A})^2 + (\frac{[A, B]}{B})^2} - \frac{1}{2}, \text{ 且要求的整数 } t_0 = \left\lceil -\frac{\frac{[A, B]}{A} * (\text{商} + x_0) + \frac{[A, B]}{B} * y_0}{(\frac{[A, B]}{A})^2 + (\frac{[A, B]}{B})^2} - \frac{1}{2} \right\rceil + 1 -$$

$$! \left(-\frac{\frac{[A, B]}{A} * (\text{商} + x_0) + \frac{[A, B]}{B} * y_0}{(\frac{[A, B]}{A})^2 + (\frac{[A, B]}{B})^2} - \frac{1}{2} - \left\lceil -\frac{\frac{[A, B]}{A} * (\text{商} + x_0) + \frac{[A, B]}{B} * y_0}{(\frac{[A, B]}{A})^2 + (\frac{[A, B]}{B})^2} - \frac{1}{2} \right\rceil \right)。$$

1. 程序: 第 10 个程序的改良版(2+5+8b)

代码如下: (vc++6.0, 899 行, 34 页 word 文档, 注明了各关键语句的含义。复制粘贴即可使用。)

```
#include<stdio.h>
```

```
#include<math.h>
```

```
#include<stdlib.h>
```

```

int getmin[2],b[2],m[2],z[2],changehappens;          /*创建一个有两元素的一维
全局数组变量 getmin[2]*/

void list(int *a1,int *a2)
{
    int min=*a1;
    changehappens=0;
    if(*a1>*a2)
    {
        *a1=*a2;
        *a2=min;
        min=*(a1+1);
        *(a1+1)=*(a2+1);
        *(a2+1)=min;
        changehappens=1;
    }
}

void minandAddress(int p2,int p22,int addressp2)
{
    getmin[0]=p2;
    getmin[1]=addressp2;
    if(p22<getmin[0])
    {
        getmin[0]=p22;
        getmin[1]=addressp2-2;
    }
}

int maxjoint(int p2,int p22)                          /*创建函数 maxjoint 调取一对值,返
回它们的最大公因数数值*/
{

```

```

int t,p[2];
p[0]=p2;
p[1]=p22;
if(p[0]<p[1])                                /*调换数据的大小顺序以便之后的
辗转相除*/
{
    t=p[0];
    p[0]=p[1];
    p[1]=t;
}
do
{
    t=p[0]%p[1];
    p[0]=p[1];
    p[1]=t;
}
while (t!=0);                                /*跳出循环后 p[0]就是*p2 和*pp2
的最大公因*/
t=p[0];
return t;
}
void get(int n,int leftovers)
{
    if(leftovers==0)
    {
    }

    else if(n==1)                            /*这个东西逻辑上最好在
leftovers==0 的后面,但事实上我们的算法允许它们交换位置.因为可能当 n==2 的时候,
进入该函数的 else 里后,发现 x[1]取 j 的最大值时,余数 leftovers-m[1]*j=0,这样会出

```


现进入下一层函数 `get(1,0)`,此时我们希望的是先判断 `leftovers` 是否等于 0,因为已经找到一个合理解了.而如果此时先判断 `n==1`,由于 `leftovers==0`,所以条件 `leftovers%m[0]==0` 恒为真,使得 `x[0]=leftovers/m[0]`;语句达到和 `for(i=0;i<n;i++) x[i]=0`;语句相同的初始化 `x[0]` 为 0 的效果,并且其他之后剩下的语句均一样.这样以另一个判断标准判断其为有效非负整数解组,让人感到有点 odd,奇怪.但从逻辑上似乎也说得通,所以在我们的算法上是行得通的.*/

```

{
    if(leftovers%m[0]==0)
    {
        b[z[0]]=leftovers/m[0];
    }
    else
        ; /*这里将不会有此 else 情况
出现:不会除不尽.*/

}
else if(m[n-1]>leftovers)
    get(n-1,leftovers); /*这个要写在 n==1 的后面,
即得先判断是否 n 减来==1.所以上次的只是特殊系数的特殊算法.因为如果出现了当
n==1 的时候 m[0]>leftovers(不知道你输入的各项系数中的最小值 m[0]是多少,而
leftovers 有可能=1,所以有可能当 n==1 的时候却进入了 m[0]>leftovers 的流程),则会
进入下一层函数,出现 m[-1],访问的空间将不在范围内.此时理应进入 n==1 的判断的.
而上一个之所以可行是因为 m[0]恒=1,而 leftovers 不会小于 1(余数难道有小余 1 的?),
所以会直接不满足 m[0]>leftovers,然后进入 n==1 的流程.*/

else
{
    b[z[n-1]]=(leftovers-leftovers%m[n-1])/m[n-1];
    get(n-1,leftovers%m[n-1]);
}
}

/*****/

```

```
int main()
```

```

{
    float t00;

    char *p1=(char*)malloc(1);

    int
    *p2=(int*)malloc(sizeof(int)),**p3,*p22,**p33,pp2[2],s,count1=0,count2=0,i,sum,k
    ,value2copy,**x,**y,*p222,*pp22,flag=1,t,a[3][2],c[2][2],x1,x2,xi1,xi2,t0;

    printf("请像这样(请不要输入 0 和非整数)输入多元一次不定方程
    a1x1+a2x2+...+anxn=b 的系数后按 enter 键:'a1,a2,a3,a4,b,':\n");

    s=getchar();                                /*创建一个 scanf 函数用于实现我
    的功能,刚开始就需要值进来,所以不用 dowhile*/

    while(s!='\n')
    {
        if(s<=57&&s>=48)
        {
            count1++;                            /*统计输入了多少个"
            假·数字"进来*/

            *(p1+count1-1)=s;                    /*将 s 的值放入距离
            p1 共(count1-1)的地址对应的内容里,即放入上一步所创建的多出来的那个空间中*/

            p1=(char*)realloc(p1,count1+1);      /*将 p1 的空间扩
            张至(假·数字个数+1)个字节,预备下一次输入。这条语句会"初始化"p1 的值*/

        }
        else if(s=='-')
            flag=-1;
        else if(s==',')
        {
            sum=0;
            /*****初始化 sum 值,为了
            下面的循环*/

            for(i=0;i<=count1-1;i++)
                sum=sum+*(p1+i)-'0')*(int)pow(10,count1-1-i);

```

```

    if(flag==-1)
    {
        sum=-sum;
        flag=1;
    }
    count1=0;p1=(char*)realloc(p1,1);
/*****初始化 count1 值和
p1 拥有的地址量,均是为了下一次循环的第一个 if*/

    count2++;                                /*统计输入了多少个"
真·数字"进来*/

    *(p2+count2-1)=sum;                      /*将 sum 的值放入
距离 p2 共(count2-1)的地址对应的内容里*/

    p2=(int*)realloc(p2,(count2+1)*sizeof(int)); /*将 p2 的空间
扩张至(真·数字个数+1)个整数空间,预备下一次输入。最后它会像 p1 一样总会空出一
个空间,不过不影响我们的继续,所以循环外不用改小一格它*/

    }
    s=getchar();
}
if(count2-1<2)
{
    printf("\n*****你输入得太少了!有解无解和解是多少都能自己算的
出来了!*****\n\n!");
    return 0;
}

    p3=(int**)malloc((count2-1)*sizeof(int*)); /*从 a0 到 an-1 的非系数
部分:x0 到 xn-1*/

    for(i=0;i<count2-1;i++)

        p3[i]=(int*)malloc(2*sizeof(int));    /*现在每个子地址只开
辟两个空间了,每对新创造的最小公倍数和右系数只构成一个二元一次不定式,两个系数,
两个非系数部分.这里只给 x 所在的非系数部分分配空间.*/

```

```

pp22=(int*)malloc((count2-1)*sizeof(int));      /*创造 pp22 来复制 p2,
保留它们的原始值,以后有用。并且要在它们的负号消失之前及时得到最原始的信息.不
能丢失此负号信息.*/

for(i=0;i<=count2-1-1;i++)

    pp22[i]=p2[i];

/*****/

for(i=0;i<=1;i++)                                /*先初始化 a0 和 a1 的系数部分
和非系数部分,这里可以沿用之前的语法,之后就不行了*/

{

    for(k=0;k<=1;k++)

        if(k==i)

            if(p2[i]>0)

                p3[i][k]=1;

            else p3[i][k]=-1;

        else p3[i][k]=0;

    p2[i]=abs(p2[i]);

}

/*****/

if(count2-1>2)

{

    p22=(int*)malloc((count2-1-2)*sizeof(int));      /*从 b0 到 bn-3 的系
数部分,这是模仿 p2 创造的,但是 p2 还包含有 b 以及一个空空间.*/

    p33=(int**)malloc((count2-1-2)*sizeof(int*));    /*从 b0 到 bn-3 的
非系数部分:y0 到 yn-3*/

    for(i=0;i<count2-1-2;i++)

        p33[i]=(int*)malloc(2*sizeof(int));          /*其子地址也只开
辟两个空间*/

}

```

```

else                                     /*城乡结合部:综合最初(k=n-3)与最
末(k=-1)二阶式子的语句,具体而微的程序雏形.*/
{
    if(p2[count2-1]%maxjoint(p2[0],p2[1])!=0)
    {
        printf("您输入的多元一次不定方程无整数解。 \n");
        free(pp22);
        pp22=NULL;
        for(i=0;i<count2-1;i++)
        {
            free(p3[i]);
            p3[i]=NULL;
        }
        free(p3);
        p3=NULL;
        free(p2);
        p2=NULL;
        free(p1);
        p1=NULL;
        return 0;
    }
    x=(int**)malloc((count2-1)*sizeof(int*));
    sum=1;
    minandAddress(p2[1],p2[0],1);
    do
    {
        pp2[0]=p2[0];                /*保留值*/
        pp2[1]=p2[1];
        if(p2[0]%getmin[0]!=0)

```

```

        p2[0]=p2[0]%getmin[0];
        if(p2[1]%getmin[0]!=0)
            p2[1]=p2[1]%getmin[0];
        if(getmin[1]==1-2)
        {
            p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
            p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
        }
        else
        {
            p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
            p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
        }
        minandAddress(p2[1],p2[0],1);
        if(getmin[0]==maxjoint(p2[1],p2[0]))
            sum=0;
    }
    while(sum);
    value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
    x[1]=(int*)malloc((count2-1)*sizeof(int));
    x[0]=(int*)malloc((count2-1)*sizeof(int));
    if(pp22[0]<0)                                /*insert 第二部分首*/
    {
        a[0][0]=-pp22[0];
        a[0][1]=-1;
    }
    else
    {
        a[0][0]=pp22[0];

```

```

        a[0][1]=1;
    }
    if(pp22[1]<0)
    {
        a[1][0]=-pp22[1];
        a[1][1]=-1;
    }
    else
    {
        a[1][0]=pp22[1];
        a[1][1]=1;
    }
    list(a[0],a[1]);
    a[1][1]=-a[1][1];
    for(i=0;i<2;i++)
        c[i][1]=a[i][1];
    /*insert 第二部分,部分尾:
保留原非系数部分的符号.*/

    if(pp22[0]*pp22[1]>0)
        /*这里才是真迹:如果系数
异号,则规定它们的最大公因数为负值.*/

        flag=maxjoint(p2[1],p2[0]);
    else
        flag=-maxjoint(p2[1],p2[0]);
    x[1][count2-2]=pp22[0]/flag;
    x[0][count2-2]=-pp22[1]/flag;
    for(i=0;i<2;i++)
        /*新的替代品__始端*/
        b[i]=0;
    for(i=0;i<2;i++)
        m[i]=p2[i];
    for(i=0;i<2;i++)

```

```

        z[i]=i;
    if(m[0]>m[1])
    {
        t=m[0];
        m[0]=m[1];
        m[1]=t;
        t=z[0];
        /*当值交换时,下角标也跟着交换.*/
        z[0]=z[1];
        z[1]=t;
    }
    if(p2[count2-1]<0)
    {
        get(2,-p2[count2-1]);
        for(i=0;i<2;i++)
            b[i]=-b[i];
    }
    else
        get(2,p2[count2-1]);
    for(i=count2-3;i>=0;i--)
    {
        x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
        x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;
        if(p2[count2-1]<0)                /*insert 第二部分,部分
首.*/

        {
            a[2][0]=-p2[count2-1];
            a[2][1]=-1;
        }

```



```

else
{
    a[2][0]=p2[count2-1];
    a[2][1]=1;
}
if(a[2][1]==-1)
{
    a[2][1]=-a[2][1];
    a[0][1]=-a[0][1];
    a[1][1]=-a[1][1];
}
/*    if(changehappens)

    printf("%d(%d*x)-%d(%d*y)=%d (变换后的方程)\n\n",a[0][0],a[0][1],a[1][0],a[1][1],a[2][0]);

    else

    printf("%d(%d*x)-%d(%d*y)=%d (变换后的方程)\n\n",a[0][0],a[0][1],a[1][0],a[1][1],a[2][0]); */

    if(changehappens)
    {
        x1=x[1][i]*a[0][1];
        x2=x[0][i]*a[1][1];
    }
    else
    {
        x1=x[0][i]*a[0][1];
        x2=x[1][i]*a[1][1];
    }

    if(x1*2<=-a[1][0]/getmin[0])                                /*新的判断标准
和对应的执行标准*/

```

```

{
    xi1=a[1][0]/getmin[0]/2-(-
x1+a[1][0]/getmin[0]/2)*(a[1][0]/getmin[0]); /*本身是-xi1=(-
x1+a[1][0]/getmin[0]/2)*(a[1][0]/getmin[0])-a[1][0]/getmin[0]/2*/
    xi2=x2+a[0][0]/getmin[0]*((-x1+a[1][0]/getmin[0]/2)-
(-x1+a[1][0]/getmin[0]/2)*(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
}
else
{
    xi1=(x1+a[1][0]/getmin[0]/2)*(a[1][0]/getmin[0])-
a[1][0]/getmin[0]/2;
    xi2=x2-a[0][0]/getmin[0]*((x1+a[1][0]/getmin[0]/2)-
(x1+a[1][0]/getmin[0]/2)*(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
    if(xi1*2==a[1][0]/getmin[0])
    {
        xi1=xi1+a[1][0]/getmin[0];
        xi2=xi2+a[0][0]/getmin[0];
    }
}

/*    printf("方程的等效随机解为(%d,%d),方程的等效下奇异解为(%d,%d),等价于
反方程的上奇异解(%d,%d).\n\n",x1,x2,xi1,xi2,-xi2,-xi1); */

t00=-
(xi1*a[1][0]/getmin[0]+xi2*a[0][0]/getmin[0])/(pow(a[1][0]/getmin[0],2)+pow(a[0
][0]/getmin[0],2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/

if(t00<0)
    t00=0;

t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的
同时转换类型,以使得分母整个为 float 型.*/

xi1=xi1+t0*a[1][0]/getmin[0];
xi2=xi2+t0*a[0][0]/getmin[0];

if(changehappens)
{

```

```

        x[0][i]=xi2/a[1][1];
        x[1][i]=xi1/a[0][1];
    }
    else
    {
        x[0][i]=xi1/a[0][1];
        x[1][i]=xi2/a[1][1];
    }
    for(t=0;t<2;t++)
        a[t][1]=c[t][1];
分尾:还原 a[0][1]和 a[1][1].*/
        a[t][1]=c[t][1];
    }
    /*新的替代品__末端*/
    printf("x%d=",1);
    for(i=count2-2;i>=1;i--)
        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
    for(k=1;k<=count2-2;k++)
    {
        printf("x%d=",k+1);
        for(i=count2-1-k;i>=1;i--)
            printf("%dt%d+",x[k][i],i);
        printf("%d",x[k][0]);
        printf("\n");
    }
    free(x[0]);
    x[0]=NULL;
    free(x[1]);
    x[1]=NULL;

```

```

free(x);
x=NULL;
free(pp22);
pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;
}
free(p3);
p3=NULL;
free(p2);
p2=NULL;
free(p1);
p1=NULL;
return 0;
}

```

```

/*****

```

p22[0]=maxjoint(p2[0],p2[1]); /*没办法,任何设计都有局限,
这里多出了一些周期之外的东西,加上上面的,有两部分周期之外的,这里除了这第一句
话,相当于下面循环中 k=0 的时候的.*/

```
p33[0][0]=1;
p33[0][1]=0;
p3[2][0]=0;
if(p2[2]>0)
    p3[2][1]=1;
else p3[2][1]=-1;
p2[2]=abs(p2[2]);
```

/*****
 *****/

```

for(k=1;k<=count2-1-3;k++)
{
    p22[k]=maxjoint(p2[k+1],p22[k-1]);

    p33[k][0]=1;                                /*p22[k]由于是上一对系数的
最大公因数,恒>0,所以直接 p33[k][0]=1,剩下的=0,以及不用取系数的绝对值了*/

    p33[k][1]=0;
    p3[k+2][0]=0;
    if(p2[k+2]>0)
        p3[k+2][1]=1;
    else p3[k+2][1]=-1;
    p2[k+2]=abs(p2[k+2]);
}

/*****/

if(p2[count2-1]%maxjoint(p2[count2-2],p22[count2-4])!=0)
{
    printf("您输入的多元一次不定方程无整数解。 \n");

    for(i=0;i<count2-1-2;i++)
    {
        free(p33[i]);
        p33[i]=NULL;
    }
    free(p33);
    p33=NULL;
    free(p22);
    p22=NULL;
    free(pp22);
    pp22=NULL;
    for(i=0;i<count2-1;i++)
    {

```

```

        free(p3[i]);
        p3[i]=NULL;
    }
    free(p3);
    p3=NULL;
    free(p2);
    p2=NULL;
    free(p1);
    p1=NULL;
    return 0;
}

x=(int**)malloc((count2-1)*sizeof(int*));
y=(int**)malloc((count2-1-2)*sizeof(int*));

p222=(int*)malloc((count2-1-2)*sizeof(int));    /*创造 p222 来复制 p22,
保留它们的原始值,以后有用,注:由于 p22 全是正的,所以 p222 也全是正的.*/

for(i=0;i<=count2-1-3;i++)
    p222[i]=p22[i];

/*****/    /*相当于 k=n-
3*/

sum=1;
minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
do
{
    pp2[0]=p22[count2-1-3];    /*保留值*/
    pp2[1]=p2[count2-1-3+2];
    if(p22[count2-1-3]%getmin[0]!=0)
        p22[count2-1-3]=p22[count2-1-3]%getmin[0];
    if(p2[count2-1-3+2]%getmin[0]!=0)
        p2[count2-1-3+2]=p2[count2-1-3+2]%getmin[0];
}

```

```

if(getmin[1]==count2-1-3)
{
    p33[count2-1-3][0]=p33[count2-1-3][0]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][0];
    p33[count2-1-3][1]=p33[count2-1-3][1]+(pp2[1]-p2[count2-
1-3+2])/getmin[0]*p3[count2-1-3+2][1];
}
else
{
    p3[count2-1-3+2][0]=p3[count2-1-3+2][0]+(pp2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][0];
    p3[count2-1-3+2][1]=p3[count2-1-3+2][1]+(pp2[0]-
p22[count2-1-3])/getmin[0]*p33[count2-1-3][1];
}
minandAddress(p2[count2-1-3+2],p22[count2-1-3],count2-1-3+2);
if(getmin[0]==maxjoint(p2[count2-1-3+2],p22[count2-1-3]))
    sum=0;
}
while(sum);
value2copy=p33[count2-1-3][0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*p3[count2-1-3+2][0];
x[count2-1-3+2]=(int*)malloc(2*sizeof(int));
y[count2-1-3]=(int*)malloc(2*sizeof(int));
if(p222[count2-1-3]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-p222[count2-1-3];
    a[0][1]=-1;
}
else
{
    a[0][0]=p222[count2-1-3];

```

```

        a[0][1]=1;
    }
    if(pp22[count2-1-3+2]<0)
    {
        a[1][0]=-pp22[count2-1-3+2];
        a[1][1]=-1;
    }
    else
    {
        a[1][0]=pp22[count2-1-3+2];
        a[1][1]=1;
    }
    list(a[0],a[1]);
    a[1][1]=-a[1][1];
    for(i=0;i<2;i++)
        c[i][1]=a[i][1];                                /*insert 第二部分,部分尾:保
留原非系数部分的符号.*/

    if(pp22[count2-1-3+2]>0)                                /*这里本来是判断
p222[count2-1-3]*pp22[count2-1-3+2]>0,但由于 p222[count2-1-3]本身就已经人为
规定>0 了,所以能让程序跑快点就让它跑快点嘛.*/

        flag=maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    else
        flag=-maxjoint(p2[count2-1-3+2],p22[count2-1-3]);
    x[count2-1-3+2][1]=p222[count2-1-3]/flag;
    y[count2-1-3][1]=-pp22[count2-1-3+2]/flag;
    for(i=0;i<2;i++)
        /*新的替代品__始端*/
        b[i]=0;
    m[0]=p22[count2-1-3];                                /*细微的差别*/

```



```

m[1]=p2[count2-1-3+2];
for(i=0;i<2;i++)
    z[i]=i;
if(m[0]>m[1])
{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;
}
if(p2[count2-1]<0)
{
    get(2,-p2[count2-1]);
    for(i=0;i<2;i++)
        b[i]=-b[i];
}
else
    get(2,p2[count2-1]);
{
    x[count2-1-3+2][0]=(p33[count2-1-3][0]*b[1]-b[0]*p3[count2-1-
3+2][0])/value2copy;
    y[count2-1-3][0]=(b[0]*p3[count2-1-3+2][1]-p33[count2-1-
3][1]*b[1])/value2copy;
    if(p2[count2-1]<0)
        /*insert 第二部分,部分
首.*/
    {
        a[2][0]=-p2[count2-1];

```

```

        a[2][1]=-1;
    }
    else
    {
        a[2][0]=p2[count2-1];
        a[2][1]=1;
    }
    if(a[2][1]==-1)
    {
        a[2][1]=-a[2][1];
        a[0][1]=-a[0][1];
        a[1][1]=-a[1][1];
    }
    if(changehappens)
    {
        x1=x[count2-1-3+2][0]*a[0][1];
        x2=y[count2-1-3][0]*a[1][1];
    }
    else
    {
        x1=y[count2-1-3][0]*a[0][1];
        x2=x[count2-1-3+2][0]*a[1][1];
    }
    if(x1*2<=-a[1][0]/getmin[0])                /*新的判断标准和对应
的执执行标准*/
    {
        xi1=a[1][0]/getmin[0]/2-(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]); /*本身是-xi1=(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-a[1][0]/getmin[0]/2*/

```

```

        xi2=x2+a[0][0]/getmin[0]*((-x1+a[1][0]/getmin[0]/2)-(-
x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
    }
    else
    {
        xi1=(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0])-
a[1][0]/getmin[0]/2;
        xi2=x2-a[0][0]/getmin[0]*((x1+a[1][0]/getmin[0]/2)-
(x1+a[1][0]/getmin[0]/2)%(a[1][0]/getmin[0]))/(a[1][0]/getmin[0]);
        if(xi1*2==a[1][0]/getmin[0])
        {
            xi1=xi1+a[1][0]/getmin[0];
            xi2=xi2+a[0][0]/getmin[0];
        }
    }
    t00=-
(xi1*a[1][0]/getmin[0]+xi2*a[0][0]/getmin[0])/(pow(a[1][0]/getmin[0],2)+pow(a[0
][0]/getmin[0],2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/
    if(t00<0)
        t00=0;

    t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的同时转
换类型,以使得分母整个为 float 型.*/

    xi1=xi1+t0*a[1][0]/getmin[0];
    xi2=xi2+t0*a[0][0]/getmin[0];
    if(changehappens)
    {
        y[count2-1-3][0]=xi2/a[1][1];
        x[count2-1-3+2][0]=xi1/a[0][1];
    }
    else
    {

```

```

y[count2-1-3][0]=xi1/a[0][1];
x[count2-1-3+2][0]=xi2/a[1][1];
}
for(t=0;t<2;t++)
    a[t][1]=c[t][1];
/*insert 第二部分,部分尾:
还原 a[0][1]和 a[1][1].*/
}
/*新的替代品__末端*/
/*****/
for(k=count2-1-3-1;k>=0;k--)
{
    sum=1;
    minandAddress(p2[k+2],p22[k],k+2);
    do
    {
        pp2[0]=p22[k];
        pp2[1]=p2[k+2];
        if(p22[k]%getmin[0]!=0)
            p22[k]=p22[k]%getmin[0];
        if(p2[k+2]%getmin[0]!=0)
            p2[k+2]=p2[k+2]%getmin[0];
        if(getmin[1]==k)
        {
            p33[k][0]=p33[k][0]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][0];
            p33[k][1]=p33[k][1]+(pp2[1]-
p2[k+2])/getmin[0]*p3[k+2][1];
        }
        else
        {

```

```

        p3[k+2][0]=p3[k+2][0]+(p2[0]-
p22[k])/getmin[0]*p33[k][0];
        p3[k+2][1]=p3[k+2][1]+(p2[0]-
p22[k])/getmin[0]*p33[k][1];
    }
    minandAddress(p2[k+2],p22[k],k+2);
    if(getmin[0]==maxjoint(p2[k+2],p22[k]))
        sum=0;
}
while(sum);
value2copy=p33[k][0]*p3[k+2][1]-p33[k][1]*p3[k+2][0];
x[k+2]=(int*)malloc((count2-1-k-1)*sizeof(int));
y[k]=(int*)malloc((count2-1-k-1)*sizeof(int));
if(p22[k]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-p22[k];
    a[0][1]=-1;
}
else
{
    a[0][0]=p22[k];
    a[0][1]=1;
}
if(pp22[k+2]<0)
{
    a[1][0]=-pp22[k+2];
    a[1][1]=-1;
}
else
{

```

```

a[1][0]=pp22[k+2];
a[1][1]=1;
}
list(a[0],a[1]);
a[1][1]=-a[1][1];
for(i=0;i<2;i++)
    c[i][1]=a[i][1];
保留原非系数部分的符号.*/
if(pp22[k+2]>0)
    flag=maxjoint(p2[k+2],p22[k]);
else
    flag=-maxjoint(p2[k+2],p22[k]);
x[k+2][count2-1-k-2]=p22[k]/flag;
y[k][count2-1-k-2]=-pp22[k+2]/flag;
m[0]=p22[k];
m[1]=p2[k+2];
for(i=0;i<2;i++)
    z[i]=i;
if(m[0]>m[1])
{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;
}
/*insert 第二部分,部分尾:
/*新的替代品__始端;细微的
差别;只有它们能写在外面;那么现在开始放在里面的许多与 i 无关的 i 得改为 t*/

```

```

for(i=count2-1-k-3;i>=0;i--)
{
    for(t=0;t<2;t++)                /*结构改变了,(从这开
始都)它跑里面来了,每次使用都需要初始化 b.*/
        b[t]=0;
    if(y[k+1][i]<0)                    /*这里本该写作
y[k+1][i]*p222[k+1]<0的*/
    {
        get(2,-p222[k+1]*y[k+1][i]);
        for(t=0;t<2;t++)
            b[t]=-b[t];
    }
    else
        get(2,p222[k+1]*y[k+1][i]);
    x[k+2][i]=(p33[k][0]*b[1]-b[0]*p3[k+2][0])/value2copy;
    y[k][i]=(b[0]*p3[k+2][1]-p33[k][1]*b[1])/value2copy;
    if(y[k+1][i]<0)                    /*insert 第二部分,部分
首.*/
    {
        a[2][0]=-y[k+1][i]*p222[k+1];
        a[2][1]=-1;
    }
    else
    {
        a[2][0]=y[k+1][i]*p222[k+1];
        a[2][1]=1;
    }
    if(a[2][1]==-1)
    {

```

```

a[2][1]=-a[2][1];
a[0][1]=-a[0][1];
a[1][1]=-a[1][1];
}
if(changehappens)
{
    x1=x[k+2][i]*a[0][1];
    x2=y[k][i]*a[1][1];
}
else
{
    x1=y[k][i]*a[0][1];
    x2=x[k+2][i]*a[1][1];
}
if(x1*2<=-a[1][0]/p222[k+1])          /*新的判断标准和
对应的执行标准*/
{
    xi1=a[1][0]/p222[k+1]/2-(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]);  /*本身是-xi1=(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1])-a[1][0]/p222[k+1]/2*/
    xi2=x2+a[0][0]/p222[k+1]*((-
x1+a[1][0]/p222[k+1]/2)-(-
x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]))/(a[1][0]/p222[k+1]);
}
else
{
    xi1=(x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1])-
a[1][0]/p222[k+1]/2;
    xi2=x2-
a[0][0]/p222[k+1]*((x1+a[1][0]/p222[k+1]/2)-
(x1+a[1][0]/p222[k+1]/2)%(a[1][0]/p222[k+1]))/(a[1][0]/p222[k+1]);
    if(xi1*2==a[1][0]/p222[k+1])

```



```

{
    xi1=xi1+a[1][0]/p222[k+1];
    xi2=xi2+a[0][0]/p222[k+1];
}
}
t00=-
(xi1*a[1][0]/p222[k+1]+xi2*a[0][0]/p222[k+1])/(pow(a[1][0]/p222[k+1],2)+pow(a
[0][0]/p222[k+1],2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/
if(t00<0)
    t00=0;

t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的
同时转换类型,以使得分母整个为 float 型.*/

xi1=xi1+t0*a[1][0]/p222[k+1];
xi2=xi2+t0*a[0][0]/p222[k+1];
if(changehappens)
{
    y[k][i]=xi2/a[1][1];
    x[k+2][i]=xi1/a[0][1];
}
else
{
    y[k][i]=xi1/a[0][1];
    x[k+2][i]=xi2/a[1][1];
}
for(t=0;t<2;t++)

    a[t][1]=c[t][1];                                /*insert 第二部分,部
分尾:还原 a[0][1]和 a[1][1].*/

}                                                    /*新的替代品__末端*/

}

```

```

/*****/ /*相当于 k=-1*/

sum=1;
minandAddress(p2[1],p2[0],1);
do
{
    pp2[0]=p2[0]; /*保留值*/
    pp2[1]=p2[1];
    if(p2[0]%getmin[0]!=0)
        p2[0]=p2[0]%getmin[0];
    if(p2[1]%getmin[0]!=0)
        p2[1]=p2[1]%getmin[0];
    if(getmin[1]==1-2)
    {
        p3[0][0]=p3[0][0]+(pp2[1]-p2[1])/getmin[0]*p3[1][0];
        p3[0][1]=p3[0][1]+(pp2[1]-p2[1])/getmin[0]*p3[1][1];
    }
    else
    {
        p3[1][0]=p3[1][0]+(pp2[0]-p2[0])/getmin[0]*p3[0][0];
        p3[1][1]=p3[1][1]+(pp2[0]-p2[0])/getmin[0]*p3[0][1];
    }
    minandAddress(p2[1],p2[0],1);
    if(getmin[0]==maxjoint(p2[1],p2[0]))
        sum=0;
}
while(sum);
value2copy=p3[0][0]*p3[1][1]-p3[0][1]*p3[1][0];
x[1]=(int*)malloc((count2-1)*sizeof(int));
x[0]=(int*)malloc((count2-1)*sizeof(int));

```

```

if(pp22[0]<0)                                /*insert 第二部分首*/
{
    a[0][0]=-pp22[0];
    a[0][1]=-1;
}
else
{
    a[0][0]=pp22[0];
    a[0][1]=1;
}
if(pp22[1]<0)
{
    a[1][0]=-pp22[1];
    a[1][1]=-1;
}
else
{
    a[1][0]=pp22[1];
    a[1][1]=1;
}
list(a[0],a[1]);
a[1][1]=-a[1][1];
for(i=0;i<2;i++)
    c[i][1]=a[i][1];                        /*insert 第二部分,部分尾:保
留原非系数部分的符号.*/

if(pp22[0]*pp22[1]>0)                        /*这里才是真迹:如果系数异号,
则规定它们的最大公因数为负值.*/

    flag=maxjoint(p2[1],p2[0]);
else

```

```

        flag=-maxjoint(p2[1],p2[0]);
x[1][count2-2]=pp22[0]/flag;
x[0][count2-2]=-pp22[1]/flag;
for(i=0;i<2;i++)
    /*新的替代品__始端;同样,只有它们能写在外面;*/
    m[i]=p2[i];
for(i=0;i<2;i++)
    z[i]=i;
if(m[0]>m[1])
{
    t=m[0];
    m[0]=m[1];
    m[1]=t;
    t=z[0];
    /*当值交换时,下角标也跟着交换.*/
    z[0]=z[1];
    z[1]=t;
}
for(i=count2-3;i>=0;i--)
{
    for(t=0;t<2;t++)
        b[t]=0;
    if(y[0][i]<0)
    {
        get(2,-p222[0]*y[0][i]);
        for(t=0;t<2;t++)
            b[t]=-b[t];
    }
    else
        get(2,p222[0]*y[0][i]);

```

```

x[1][i]=(p3[0][0]*b[1]-b[0]*p3[1][0])/value2copy;
x[0][i]=(b[0]*p3[1][1]-p3[0][1]*b[1])/value2copy;

if(y[0][i]<0)                                     /*insert 第二部分,部分首.*/
{
    a[2][0]=-y[0][i]*p222[0];
    a[2][1]=-1;
}
else
{
    a[2][0]=y[0][i]*p222[0];
    a[2][1]=1;
}
if(a[2][1]==-1)
{
    a[2][1]=-a[2][1];
    a[0][1]=-a[0][1];
    a[1][1]=-a[1][1];
}
if(changehappens)
{
    x1=x[1][i]*a[0][1];
    x2=x[0][i]*a[1][1];
}
else
{
    x1=x[0][i]*a[0][1];
    x2=x[1][i]*a[1][1];
}

```

```

if(x1*2<=-a[1][0]/p222[0])                                /*新的判断标准和对应
的執行标准*/
{
    xi1=a[1][0]/p222[0]/2-(-
x1+a[1][0]/p222[0]/2)*(a[1][0]/p222[0]); /*本身是-xi1=(-
x1+a[1][0]/p222[0]/2)*(a[1][0]/p222[0])-a[1][0]/p222[0]/2*/
    xi2=x2+a[0][0]/p222[0]*((-x1+a[1][0]/p222[0]/2)-(-
x1+a[1][0]/p222[0]/2)*(a[1][0]/p222[0]))/(a[1][0]/p222[0]);
}
else
{
    xi1=(x1+a[1][0]/p222[0]/2)*(a[1][0]/p222[0])-
a[1][0]/p222[0]/2;
    xi2=x2-a[0][0]/p222[0]*((x1+a[1][0]/p222[0]/2)-
(x1+a[1][0]/p222[0]/2)*(a[1][0]/p222[0]))/(a[1][0]/p222[0]);
    if(xi1*2==a[1][0]/p222[0])
    {
        xi1=xi1+a[1][0]/p222[0];
        xi2=xi2+a[0][0]/p222[0];
    }
}
t00=-
(xi1*a[1][0]/p222[0]+xi2*a[0][0]/p222[0])/(pow(a[1][0]/p222[0],2)+pow(a[0][0]/p
222[0],2))-0.5; /*t00 是 float 型,这里不能将它赋值给整型的 t0.*/
if(t00<0)
    t00=0;
t0=(int)t00+1-!(t00-(int)t00); /*我在分母处加了 pow 运算的同时转
换类型,以使得分母整个为 float 型.*/
xi1=xi1+t0*a[1][0]/p222[0];
xi2=xi2+t0*a[0][0]/p222[0];
if(changehappens)
{

```

```

        x[0][i]=xi2/a[1][1];
        x[1][i]=xi1/a[0][1];
    }
    else
    {
        x[0][i]=xi1/a[0][1];
        x[1][i]=xi2/a[1][1];
    }
    for(t=0;t<2;t++)
        a[t][1]=c[t][1];
    还原 a[0][1]和 a[1][1].*/
    }
    /*新的替代品__末端*/
/*****/
    printf("x%d=",1);
    for(i=count2-2;i>=1;i--)
        printf("%dt%d+",x[0][i],i);
    printf("%d",x[0][0]);
    printf("\n");
    for(k=1;k<=count2-2;k++)
    {
        printf("x%d=",k+1);
        for(i=count2-1-k;i>=1;i--)
            printf("%dt%d+",x[k][i],i);
        printf("%d",x[k][0]);
        printf("\n");
    }
/*****/
    free(x[0]);
    x[0]=NULL;

```

/*insert 第二部分,部分尾:

/*新的替代品__末端*/

/*****/

/*****/

```
free(x[1]);
x[1]=NULL;
for(k=0;k<=count2-1-3-1;k++)
{
    free(y[k]);
    y[k]=NULL;
    free(x[k+2]);
    x[k+2]=NULL;
}
free(y[count2-1-3]);
y[count2-1-3]=NULL;
free(x[count2-1-3+2]);
x[count2-1-3+2]=NULL;
free(p222);
p222=NULL;
free(y);
y=NULL;
free(x);
x=NULL;
for(i=0;i<count2-1-2;i++)
{
    free(p33[i]);
    p33[i]=NULL;
}
free(p33);
p33=NULL;
free(p22);
p22=NULL;
free(pp22);
```



```

pp22=NULL;
for(i=0;i<count2-1;i++)
{
    free(p3[i]);
    p3[i]=NULL;
}
free(p3);
p3=NULL;
free(p2);
p2=NULL;
free(p1);
p1=NULL;
}

/*pp22 是 p2 的复制品,保留了它的符号,是纯 a 类系数;然后 p2 符号一点点地全为正了,
且用 pp2 记录了 p2 和 p2、p2 和 pp2 的最大公因数的 b 类系数;p222 是 p22 的复制品,
即是一系列 b 类系数和最大公因数;p2 与 p2、p2 与 p22 之后会进行超级辗转相除,丢
失信息,pp22 和 p222 的价值便体现出来了:原始的二元一次不定方程们的 3 个系数.*/

/*p222[k]x+pp22[k+2]y=y[k+1][i]*p222[k+1]*/
/*p22[k]x+p2[k+2]y=|y[k+1][i]*p222[k+1]|*/

/*m[z[1]]是超级辗转相除后的 p2 与 p2、p2 与 p22 中的较大的系数,b[z[1]]是它的非
系数部分,即超级辗转后的全正系数方程的解.*/

```

后记

如果你通过 IV.以及 III.(或者说第 12、第 11 这倒数的两个程序)输入同一串数字,你会发现在大多数情况【高维下,虽然尝试成功的次数多了,但是邻近解的个数也多了】下它们的得到的结果是完全相同的(特别是方程的元数大于 2 的时候),尽管

在关键地方采用的是完全不同的算法。这说明很多时候，在高维度下，绝对值意义上的最小整数解就是平方和意义上的最小整数解，and 平方和意义上的最小整数解就是绝对值意义上的最小整数解。(但这其实只是我的程序算法得出的结果有这个暗示信息而已，它们是否真正是这样，尚不得而知)——你可以通过同一串数字在两个程序中的输出结果的比较，来检验其中一个程序是否得到了比另一个程序“在另一个程序的最小整数解意义上的”更小的整数解，以检验另一个程序是不是在说谎。——即你可以用这种方法互相检验对方是否得到了对方意义上的最小整数解，进而检验程序 or 理论的正确性。

最后两个程序 or 理论并不能百分之百地求得对应意义上的最小整数解【但确实是相对而言能得到较小解和(用于求解的)系数的较为优秀的算法吧。。】，比如若人为地创造“ $23, x_2, x_3, x_4, x_5, x_6, \dots, x_k, -23,$ ”这样的，故意刁难程序的系数数列，则你放心，程序会被你难倒的：此时在各种意义上的最小整数解应该是 $(-1, 0, 0, 0, 0, \dots, 0)$ ，然而程序只有在当 23 离 -23 更近的时候：比如“ $x_1, x_2, x_3, x_4, x_5, x_6, \dots, 23, -23,$ ”这样，才更有可能得到符合期望的结果： $(0, 0, 0, 0, 0, \dots, -1)$ 。——虽然我一直认为，若一个问题有解的话，一定有该解所对应的解的方法，即有关于通向该解的方法的解，即使关于解它的方法的解不止一个——一个“因目标情况的具体数值 x 的变化而变化”的映射结果 $f(x)$ 是不可怕的，但一个“因目标情况的具体数值的变化而变化”的映射 f ，却是非常令人头大的。

或许我们的问题就属于：不仅 $f(x)$ 随着 x 在变化，而且 f 也随着 x 在变化的问题吧——不过万一 x 到 x' 的过程中， f 到 f' 的映射 $f' = g(f)$ 中的 g 不变化呢(像加速度一样(恰好是重力加速度的符号哈。。))。。【我不认为这里的 g 是“遍历”这种“求方法的方法”哈！——由于“遍历”可以是 f ，可以是 g ，即 f 的 f ，当然也可以是 f 的 f 的 f ，即 g 的 f ；那么因为它什么都可以是，所以“遍历”也可以什么都不是】看来向更高维扩展是人类唯一的自我安慰了：它是逻辑的，因有可行性但目前不可行而成为了不可否认但也不可肯定的——但我们只需要“它的不可否认”的特点就够了，足以让我们自我安慰很久了。。。