基于矩阵的化学方程式配平的编程实现

内容摘要： 通过对化学方程式配平的数学本质的研究，设计出相关算法并使用编程语言实现。分析算法实现的复杂度与优点不足。

关键词： 矩阵消元法 化学方程式配平 编程

一、分析问题

在化学中，化学反应是遵循质量守恒定律的。可以知道，化学反应前后原子种类、原子数目、元素种类等等都是不发生改变的。因此，配平后的化学方程式两边同一种原子满足等号两边总数量相等。根据这一结论，可以尝试用"待定系数法"来配平化学方程式。

具体操作时，对于化学方程式中的每一项物质都设一个未知数，然后根据原子守恒列出方程组求解。例如方程式 $NaAlO_2 + CO_2 + H_2O == Al(OH)_3 + Na_2CO_3$（未配平）。分别将 $NaAlO_2$,CO_2,H_2O,$Al(OH)_3$,Na_2CO_3 的系数设为 x_1,x_2,x_3,x_4,x_5，根据质量守恒定律，有如下方程组（括号里为依据）：

$$\begin{cases} x_1 = 2x_5 (Na\text{原子守恒}) \\ x_1 = x_4 (Al\text{原子守恒}) \\ 2x_1 + 2x_2 + x_3 = 3x_4 + 3x_5 (O\text{原子守恒}) \\ 2x_3 = 3x_4 (H\text{原子守恒}) \end{cases}$$

直观上看，这个方程组中有 5 个未知数，却只有 4 个方程，无法确定一组唯一的解（相关定理见下文）。因为如果反应确定的话，参与反应的物质只需成一种固定比例关系，在化学方程式配平最终的结果中才要求将系数化简。因而，可以设 $x_5 = 1$，容易解得：

$$\begin{cases} x_1 = 2 \\ x_2 = 1 \\ x_3 = 3 \\ x_4 = 2 \\ x_5 = 1 \end{cases}$$

这已经是最简形式。根据化学知识，这组解确实符合原方程式配平的结果。

除了 x_5，还可以设 $x_4 = 1$，则有：

$$\begin{cases} x_1 = 1 \\ x_2 = \frac{1}{2} \\ x_3 = \frac{3}{2} \\ x_4 = 1 \\ x_5 = \frac{1}{2} \end{cases}$$

这组解的比例关系与设 $x_5 = 1$ 时的解的比例关系是相同的，也就是说这两个解化学意义上是相同的。但在配平中，就要将后者这个解化简到最简整数比。此时，要对每个解乘 2，达到化简的目的。

观察这个例子，可以发现对化学方程式的配平，相当于解一个线性方程组。这也是其数学本质。设方程式有 m 项物质，n 种元素，每项系数分别为 x_1,x_2,x_3,$\cdots$,x_m。一般的，有：

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \cdots + a_{1,m}x_m = 0 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \cdots + a_{2,m}x_m = 0 \\ \quad\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + \cdots + a_{n,m}x_m = 0 \end{cases} \tag{1}$$

$a_{i,j}$ 表示第 i 种原子在第 j 项物质中的数量，在化学方程式等号左边时为正，在等号右边时为负，这样才能满足原子守恒。而最终配平后的每项物质系数就是解得的的对应的 x 的值。

既然是线性方程组，那么可以借助线性代数中矩阵的相关知识。将以上方程组写成矩阵形式，那么即为：

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2)$$

根据数学知识，(1) 和 (2) 是等价的。

二、设计算法

(一)定义和定理

1.矩阵[1]

在数学中，矩阵（Matrix）是一个按照长方阵列排列的复数或实数集合，最早来自于方程组的系数及常数所构成的方阵。

定义： 由 $n \times m$ 个数 $a_{i,j}$ 排成的 n 行 m 列的数表称为 n 行 m 列的矩阵，简称 $n \times m$ 矩阵。矩阵 A 记作：

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix}$$

这 $n \times m$ 个数称为矩阵 A 的元素，简称为元。主元定义为：主元是矩阵每个非零行第一个非零元素。

2.消元法和行变换

定义： 消元法是指将许多关系式中的若干个元素通过有限次地变换，消去其中的某些元素，从而使问题获得解决的一种解题方法。

消元法理论的核心主要如下：

- 两方程互换，解不变；
- 一方程乘以非零数 k，解不变；
- 一方程加上另一方程，解不变；[2]

体现在矩阵中，以上变换分别是：

- 矩阵的某两行交换；
- 矩阵的某一行乘以非零数 k；
- 矩阵的某一行加上另一行；

这些也是矩阵的初等行变换。

消元法将方程组中的一方程的未知数用含有另一未知数的代数式表示，并将其代入到另一方程中，这就消去了一未知数，得到一解；或将方程组中的一方程先倍乘某个常数再加到另外一方程中去，也可达到消去一未知数的目的。简单来说，消元法就是通过初等变换尽可能多的消除未知量。

行等价定义为：若矩阵 A 和矩阵 B 是行等价的，就是说 A 经过若干次初等行变换可以变成 B。

3.高斯消元法

在求解线性方程组中，高斯消元法有着重要地位。其首先将方程的增广矩阵（增广矩阵即将系数矩阵与常数矩阵的增广）利用行初等变换化为行最简形，然后以线性无关为准则对自由未知量赋值，最后列出表达方程组通解。它主要有五个步骤：[3]

1. 增广矩阵行初等行变换为行最简形；
2. 还原线性方程组；
3. 求解第一个变量；
4. 补充自由未知量；
5. 列表示方程组通解。

高斯消元法还存在另一个版本：高斯-约旦消元法（或译作高斯-约尔当消元法）。不同所在是其最终将矩阵的每一行都变成只有一项系数，但此算法效率较低。[4]

4.一些结论

1.线性方程组有解的充要条件是增广矩阵的最右列不是主元列，即增广矩阵的阶梯形没有形如$[0\ \cdots\ 0\ a]$,$a\neq 0$的行。若有解，则有两种情形：有唯一解；有无穷多解。

2.若两个矩阵是行等价的，则它们具有相同的解集。

根据第一个结论，可以判断解的存在性；根据第二个结论，可以通过矩阵的初等行变换求解未知数。

本文所涉及的定义、结论都是较为简单基础的，读者可在国内外大多数线性代数教科书上找到相关内容，本文只给出一些有关的定义。本文主要参考David C. Lay等著刘深泉等译的《线性代数及其应用(第5版)》（北京机械工业出版社），不再一一列出此相关参考文献。

5.本文符号注记

本文中，$\gcd(a,b)$指a和b的最大公因数；$\mathrm{lcm}(a,b)$指a和b的最小公倍数；$\min(a,b)$指a和b中的较小值。

（二）消元算法

尽管存在现成的解线性方程组的数学算法（上文提到的高斯消元法）。但在本题目中，算法是要基于化学原理的，这就对算法的设计提出了更高的要求。

首先，考虑对输入的处理。假设输入的方程式已转化为**问题分析**中的矩阵形式(2)。

观察这个矩阵的形式，可以看到常数矩阵的每项系数都是0，其实可以不必按照这个的矩阵求解。化学方程式各项系数是等比例发生改变的，那么可以将某一项的未知系数直接设为1，再将它作为常数项而转移到常数矩阵中。那么要求解的方程式就发生了改变：

$$Ax=B\iff\begin{bmatrix}a_{1,1}&a_{1,2}&\cdots&a_{1,m-1}\\a_{2,1}&a_{2,2}&\cdots&a_{2,m-1}\\\vdots&\vdots&\ddots&\vdots\\a_{n,1}&a_{n,2}&\cdots&a_{n,m-1}\end{bmatrix}\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}=\begin{bmatrix}-a_{1,m}\\-a_{2,m}\\\vdots\\-a_{n,m}\end{bmatrix}(3)$$

这样做将一个齐次线性方程组转化成了一个非齐次线性方程组，这样做是不影响求解的，(2)和(3)的解在化学方程式配平下是等价的，将在下文介绍。

高斯-约旦消元法最终的结果矩阵虽然很简洁，但是这样的结果是基于方阵运算的（在本问题中，对于(3)式，$n=m-1$），否则，最终给出的结果并不能满足矩阵的每一行都变成只有一项有系数。但是通过高斯-约旦消元法得到的结果矩阵，相对于高斯消元法得到的结果矩阵，在处理最终答案上，会更加简单明了。所以，本算法主要参考高斯-约旦消元法的算法思想，尽可能将矩阵变换到最简。

对于(3)，根据这些认识，可以设计出如下算法步骤：

1. 将方程式 (3) 写成增广矩阵形式。

$$A|B=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m-1} & -a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m-1} & -a_{2,m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m-1} & -a_{n,m} \end{bmatrix} (4)$$

2. 消元：对矩阵进行初等行变换以消元。初等行变换的内容已在上文介绍过了。具体操作是：顺序枚举矩阵每一行 i，通过同乘或同除一个非零实数将其第 i 列的系数转化为 1。接着枚举第 1 行到第 n 行行 j，通过 $a_{i,i}$ 与 $a_{j,i}$ 之间的关系，通过加减消元将每一行第 i 列系数全部消掉。在这个操作之前，当前行的第 1 个到第 $\min(i,n)-1$ 个的元素都已经变成 0 了。

3. 对于结果矩阵，进行解的判断和给出。一般来说，如果是方阵的话（即 $n=m-1$），它主对角线全为 1，左下三角矩阵元素和右上矩阵元素（除了第 m 列）全为 0。形式如下：

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 & a'_{1,m} \\ 0 & 1 & 0 & \cdots & \cdots & 0 & a'_{2,m} \\ 0 & 0 & 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 & 0 & a'_{n-1,m} \\ 0 & 0 & 0 & \cdots & 0 & 1 & a'_{n,m} \end{bmatrix} (5)$$

否则，它除了左下矩阵元素必为 0，甚至不能保证主对角线上元素为 1（可能存在为 0 的情况）。形式大体如下：

$$[A'B']=\begin{bmatrix} a'_{1,1} & a'_{1,2} & a'_{1,3} & \cdots & \cdots & a'_{1,m} \\ 0 & a'_{2,2} & a'_{2,3} & \cdots & \cdots & a'_{2,m} \\ 0 & 0 & a'_{3,3} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & a'_{n-1,m-1} & a'_{n-1,m} \\ 0 & 0 & 0 & \cdots & 0 & a'_{n,m} \end{bmatrix} (6)$$

消元算法的大体内容就到了这里，看似简单但在具体实现上还有许多值得注意的地方。

（三）解的判断与给出

对于具体问题上来说，最终形成的矩阵的大小是 $n\times m$ 的（包括常数矩阵），这便对高斯消元（一般适用于 $n\times n$ 的方阵）的解的判断提出了更高的要求。

首先，存在解是多解的必要条件，所以，在解的判断上，要先判断解的存在性。可以首先对数学方程式进行判断，先依据上文提到的结论，即矩阵中不能存在形如的 $[0\ \cdots\ 0\ a],a\neq 0$ 的某一行，其数学意义是常量不能等于 0。在化学方程式配平的意义下的，所以解的存在还需符合解不为负数或 0。注意到可能存在一个未知量决定于其他的未知量，导致这个未知量在回带过程中会出现等于 0 或为负数的情况，这一点的避免将在下文给出，此时暂未执行解的给出的相关操作，所以还不影响解的判断。现在只要判断是否存在恒等于负数或 0 的未知数即可。对于解恒为 0 的情况，可以先对化学方程式等号两边的元素种类进行检查，如果存在一种元素，只在一边出现，那么这个化学方程式配平必无解。还有就是判断矩阵中只含一个未知数的行，此时这个元的值恒等于第 m 列上的数，这是清楚的。上文中提到，将齐次线性方程组转换成非齐次线性方程组不影响解的判断，原因在于解是等比例变换的。以齐次线性方程组形式的话，某些情况下，可以解都为 0，但这在化学配平意义下是无解的。如果先设一个未知数为 1，那么消元过程中，各解必不能成比例存在，即会出现 $[0\ \cdots\ 0\ a],a\neq 0$ 的行，此时矩阵无解。这些就是解的存在性的判断。

考虑多解的判断。只要存在解且不符合 (5) 式，那必然就是多解了，因为此时还是存在自由变量等待赋值。

解的存在性判断完了，可以执行解的给出。对于形如 (5) 的矩阵，可以直接得出答案。答案即为：

$$x = \begin{bmatrix} a'_{1,m} \\ a'_{2,m} \\ \vdots \\ a'_{n,m} \end{bmatrix}$$

对于不符合方阵形式的矩阵 (6)，在接下来的操作中，要确定自由的未知数的值，通过回带，得出答案。因为解的存在性已经判断完了，现在只需尝试给出一组特解即可，特别注意不能让回带后的其他未知数为负数和 0 即可。

具体编程实现中，可以考虑到递归的思想。因为矩阵的数学意义，矩阵的每一行都是未知数之间的关系。可以在消元过程中询问矩阵每行的未知数的数量，如果只有一个，那么这个未知数就恒为第 m 列上的数，然后回带，将每一行的这个未知数移项消去。如果未知数的数量大于 1，那么就需要在之后的过程中去通过确定一些未知数的值，来减少未知数的数量。所以，对于有 2 个及以上未知数的行，确定了其中一个未知数后，矩阵每一含有这个未知数的行都要做移项处理，然后再判断这一行是否只剩一个未知数，如果是，可以直接确定这个未知数的值（已经可以算是常量了）。同上述一样确定这个未知数后递归处理这个未知数，直到不能求解了。然后再通过确定未知数，重复这个过程。

那么怎么开始这个过程呢？可以先把所有的固定的未知数的值求出来，再反复回带。如果还存在未知数，那么就找出未知数数量最少的那一行，确定其中的未知数的值。这个值又要怎么确定呢？考虑到多解的情况，有些未知数可能在回带过程中出现“恒”为负数和 0 的情况，这显然不是所希望看到的。那么可以枚举一定范围内的实数，再通过判断这个未知数所直接影响的其他未知数的值（矩阵的某一行只有两个未知数，且其中一个为当前将要确定值的未知数，另一个未知数便是所直接影响的其他未知数），如果符合为正数，那么即可。

（四）选主消元

选主消元意为在每次进行操作时选择适当的主元所在的行进行行变换，而不是直接对每次顺序枚举到的行进行行变换，这是为了避免精度所带来的结果偏差。假设当前主元列为第 k 列，主元行为第 cur 行，$a_{cur,k}$ 还未化成 1，当前对第 i 行进行行变换，并进行到第 j 列。那么，元素 $a_{i,j}$ 此时应该改变为：$a_{i,j} = a_{i,j} - \dfrac{a_{i,k}}{a_{cur,k}} \times a_{cur,j}$，$a_{i,k}$ 的值不是我们所能决定的。但观察这个式子，$a_{cur,k}$ 越大，$\dfrac{a_{i,k}}{a_{cur,k}}$ 越小。这个 $\dfrac{a_{i,k}}{a_{cur,k}}$ 越小，也就越逼近 0，小数点后的精确位数会更长，那么对于问题的求解来说，这样减出来的值误差越小。而且，如果 $a_{cur,k}$ 过小，分数的结果也很可能出现溢出的情况。

在计算机内部，对于浮点值，都是用有限的数字（C++中，常规的浮点值最多也只能精确到小数点后十几位）来逼近无限小数。因为这个缘故，所以在配平计算中如果不选择好主元，那么最终的结果将是失之毫厘谬以千里，对正确的答案的输出及其不利。所以，要在每次进行操作时选择恰当的行。而这行的主元应该是这个主元列上绝对值最大的。

具体操作是：顺序枚举矩阵每一行 i 时，每次选出一个第 i 行到第 n 行中某一行 cur 行，这个 cur 行的第 i 列的值是这一列上的最大值，将这个第 cur 行与当前行交换，继续上文中接下来的步骤。

（五）其他注意点

可以注意到在枚举矩阵每一行 i 时，从第 i 行到第 n 行的第 i 列上的值可能都为 0，这时程序可能就会给出错误的判断了。此时第i列上的值都为 0，按照操作，此时的行变换将不对矩阵产生改变，进行下一步时还跳过了第 i 行。所以具体在枚举矩阵的行与列时，行数与列数是不能绑定的，两者应该分开，顺序枚举的应该是列，而对于行数则另外用变量标记。对于可以找到符合主元值不为 0 的第 cur 行，那么可以在操作后将行数与列数均加一，继续下一步操作。否则，只在列数加一，行数不变。本质上，这还是通过选取恰当的主元再进行矩阵的变换。

但是，还是可以注意到，消元过程中是将第 1 行到第 n 行都有消去操作，是不是可以认为目前跳过了第 i 行也没问题？可以看到，在选主消元中，是在当前行到最后一行寻找恰当的行来操作，所以还是不能直接跳过任意一行的。

因为小数类型的运算存在误差，所以有的本该为 0 的值，其会变成一个非常接近 0 的浮点数，小数点后可能保留相当长的数字，在编程过程中，要特别注意这种情况。所以要设置一个误差值，在误差值以内，则可直接近似为 0。

三、编写程序

主体算法的设计讨论已经完成，即可开始编写程序代码。

(一)输入

为了最大量简化手工操作，使得只需输入一个方程式便能完成余下步骤，而不是在程序外先手算矩阵再输入程序求解（这显然违背了我们利用计算机降低人工成本的本意），需要对输入的方程式进行处理来构造方程组（矩阵）。那么，就要对方程式的每一项（每一种物质），都要求出其中每个原子的个数，作为未知数系数添加到待求解矩阵中去。因为存在括号等，对于方程式的处理，可以利用栈或递归思想。

如果采用递归，可以将每个括号作为一层递归来处理，栈的话同理。统计时，统计字母与数字。统计字母时，一个化学元素的表示是首位是大写的英文字母接上小写的英文字母。这些操作尚比较简单，不加赘述。在此过程中，应该特别注意括号的配对问题。例如对物质 $[Cr(N_2H_4CO)_6]_4[Cr(CN)]_3$（一种尿素铬配合物）的处理，不能直接从后往前查找右括号，因为括号的配对不同。

（二）算法执行

由输入操作之后，增广矩阵已经建立起来了，直接应用上述算法对待求解矩阵进行处理。

（三）输出

由于直接进行算法执行，最后得出的答案表示将是小数或分数（使用分数类封装的话）。这就要求对结果进行化简。假设这些答案可以表示为 $\{\frac{a_1}{b_1},\frac{a_2}{b_2},\frac{a_3}{b_3},\dots,\frac{a_m}{b_m}\}(\gcd(a_i,b_i)=1)$，那么就是要求出 $\frac{\operatorname{lcm}(b_i)}{\gcd(a_i)}$，然后将所有的答案同时乘以这个值。考虑到如果使用浮点型数值表示答案，因为大多数的化学方程式的系数还是相对较小的，可以在一定范围内枚举每一个数，使得它尽可能接近 $\frac{\operatorname{lcm}(b_i)}{\gcd(a_i)}$。怎么判断哪个数更接近这个结果呢？将结果乘以这个数将会得到一个商，将商与这个商的取整后的数进行比较，如果差值小于一定值，那么就可以认为这个商是可能的最终的答案系数。这样对所有的系数进行这个操作，如果都符合这样的预期，那么就将所有处理后的商输出，而当前的这个数也是最符合条件的数。那么有没有可能这样的输出会使得最终的答案没有化到最简？如果待处理的结果已经带有大于一的最大公因数，通过枚举小数，可以将这个最大公因数消去。而在枚举这个数的过程中，会不会导致乘完后的商仍带有大于一的最大公因数？假设枚举的这个数为 p，最大公因数为 d。如果是原先的答案就带有 d，那么 p 取 $p=\frac{1}{d}\times k$ 时，可以消去这个 d。而假设是乘完后的商带有 d，那么必是在枚举 p 时候引入的。即 $p=d\times p^{'}$。当 $d<1$时，显然对当前问题无意义，因为这可以归到前面那种情况中；当 $d>1$ 时，$p^{'}<p$，已经先被枚举过了，作为最终答案而输出了。所以顺序枚举这个数，不会使答案出现偏差。

考虑精度问题，只要将允许的误差设置的足够小，就已经可以很大减少误差了。但是，通过枚举来寻找这个数也有一个问题，就是可能在过小范围或过长步长内，无法找出满足要求的数或是偏差很大；如果查找的范围过大或步长过小，那么时间开销就很大，当然，这些是在实数表示下产生的问题。

化学方程式中，若物质系数为 1，则这个 1 省略而不输出，同时要注意上文所说的精度误差问题。

（四）数据结构

因为在这个消元算法中，只有改变矩阵的行与行之间的相关位置、元素的大小，不必使用矩阵加法、矩阵乘法等操作，故无需手写强大的矩阵类，甚至可以直接使用数组替代。但为了方便于输入输出和维护，也可以编写一个含有输入输出操作的矩阵类，通过对运算符的重载，可以极大减轻编写代码过程中的调试工作的强度。

出于对精度的要求，小数（浮点型数值）应该使用分数类替代。即在操作过程中，全部是对元素的分数形式进行操作。但在编写过程中，对于一些化学方程式，频繁的通分使得部分分母的数值在很短时间内就超过了 2^{63}（约等于 9×10^{18}），过大的分母超出了这个编程语言一般情况下所允许的最大值。这就需要引入基于字符串操作的高精度运算，需要编写一个大整数类。

但是，基于字符串操作的高精度运算频繁的对长数组进行操作，频繁地加法操作、乘法操作、取余操作将制造极大的时间开销。考虑到之前算法设计中，通过消元过程中的选主消元，已经可以极大减少精度所带来的问题。权衡之下，源码仍是采用小数形式操作。

(五)UI界面

为了引导使用者更好利用此程序，我们为此基于cmd命令行的程序编写了一定的UI界面。出于方便考虑，界面文本等完全写入源代码。因为不同编译器可能使用不同的中文字符集，易造成乱码情况。因此，文本全部使用英文编写。

综上所述，就实现了基于矩阵的化学方程式配平的编程。

四.程序分析

（一）时间复杂度

假设待求解方程式的原子种类有 n 种，共有 m 项物质，即待求解矩阵的大小为 $n \times m$。根据上文讨论，则选择主元时比较次数为 $\sum_{i=1}^{n}(n-i)$，行交换最大开销为 $m \times n$ 次交换运算，每行乘除主元系数化为 1 共乘法、除法操作 $\sum_{i=1}^{n}(m-i+1)$ 次，每次消除其它行主元列系数最大开销共 $n \times \sum_{i=1}^{m}(m-i+1)$ 次。将这些相加，此主体算法时间复杂度约为：$\mathrm{O}(n \times m^2)$，n 为元素种类数，m 为物质的数量。朴素的高斯消元法时间复杂度为 $\Theta(n^3)$。[5]

直观上来看，消元部分代码中最多存在三重循环，也可以认为时间复杂度为$\mathrm{O}(n \times m^2)$。

考虑到复杂度的常数，如果使用高精度操作，这个常数无疑将会很大，也就是说时间开销会远超想象。这就是选择舍弃一部分精度而使用浮点形式的原因。

（二）空间复杂度

空间开销主要由 $n \times m$ 大小的数组贡献。空间复杂度：$\Omega(n \times m)$，n 为元素种类数，m 为物质的数量。

（三）优点与不足

21世纪，大数据、人工智能的广泛应用，极大丰富和便利了人们的学习工作生活。在科学领域，计算机技术正日新月异地助推研究活动。有人说，“二十一世纪是化学的世纪”，还有的人说，“二十一世纪是计算机的世界”。利用计算机科学技术与其它学科交叉，已成为一种流行的研究方法。

而在之前研究的基础上，对比前人的工作，本文主要有如下创新与突破。

1.实现了无解的准确判断与多解特解的准确给出：这两点是许多开源代码的缺陷。它们只完全依靠数学公式定理，不能充分考虑到化学方面的一些因素，所以其代码对于一些特定的反应，特别是氧化还原反应，往往不能正确的给出判断，其多解的特解答案也往往出现系数为0或负数的情况。而本文则通过应用递归、枚举、反悔操作等算法思想、元素守恒等化学思想，实现了较为完善的操作。唯一的不足仅仅是特解的系数可能会相对过大，但结果仍是正确的。

2.较为创造性地实现了消元算法。尽管高斯（-约旦）消元法已有了一定的相关的计算机科学研究。但是，其他人的研究往往基于教学、纯数学方面的思考，而消元法与化学相关的研究又较为缺少，特别是缺少完善的编程实现。本文则通过高斯-约旦消元法形式的算法过程实现，高斯消元法形式的解的判断，解决了第一点中提到的一些问题，较为创新地实现了消元算法。

3.对此课题做了较系统的总结。在研究的过程中，我们发现过去是存在这个课题的研究与开源代码的。但前者往往只从数学角度出发，过于抽象；后者对于问题的研究没有深入说明，开源代码也存在许多不足。而我们完成的源码与论文，将是很好的对化学方程式配平在计算机上结合的一个总结。详尽的论文与源码将会更方便其他人的学习研究。

当然，除了上述中特解的系数相对过大以外，还存在着一些不足。针对源代码来说，代码编程者缺乏工业开发经验，是基于算法竞赛经验上编写的相关代码。所以，代码在可读性、可维护性上存在一定缺失。但是，我们又最大限度地使得代码清晰易懂、简单明了，在变量名、函数名等上遵循了一定的统一的规则。对于编译运行结果来说，源码是基于cmd命令行编写，所以程序的可操作空间仅仅限于一个小黑框和键盘键入。但是我们又用较为详尽的英文说明界面，使得使用者能更好使用我们的程序。

五.总结

本论文通过对化学方程式配平的数学本质的探讨，设计出相关算法并使用编程语言实现。对于编程过程中出现的很多细节性的问题，本论文也对其加以分析讨论。最终的代码已经足以写出，具体可以参考附件。对于这个编程实现的优劣，也已经进行了探讨。同时欢迎各位读者指出不足！

附录中将提供程序规范与源码。

六、附录

A.程序文档

> 这个程序只在解决问题钱考虑一定的输入错误的情况，较少考虑鲁棒性，故对于数据的输入有一定要求。
>
> ### 一、UI
>
> 这个程序有一定的UI界面，在UI界面有一定程度考虑输入情况。但注意：如果程序要求输入数字，则不能输入非数字字符，否则很有可能陷入死循环。对于输入的数字，不应该超出$\pm 2^{31}-1$。
>
> ### 二、方程式输入
>
> 对于方程式的输入，按照程序算法，除了字母以外，其它的都不应该出现错误。例如，不能多键入"("、")"、"+"、"="。"("和")"要求在同种物质中一一配对，否则可能出现未知错误。除了52个大小写英文字母、10个数字、上述4种符号（注意：都是在英文条件下输入），不应该出现其它字符。
>
> 例：对于$NaAlO_2+CO_2+H_2O=Al(OH)_3+Na_2CO_3$，
>
> ①$NaAlO_2+COO+H_2O=Al(OH)_3+Na_2CO_3$，是被允许的。因为$COO$与$CO_2$意义相同。
>
> ②$NaAlO_2+CO_2+H_2O=AlOH_3+Na_2CO_3$，是被允许的。尽管$AlOH_3$的物质输入是错误，但程序仅会给出不能符合预期的答案而已。
>
> ③$NaAlO_2+CO_2+H_2O=Al((OH))_3+Na_2CO_3$，是被允许的。尽管有括号嵌套，但是括号是一一配对的。
>
> ④$NaAlO_2+CO_2+H_2O=Al((O)H)_3+Na_2CO_3$，是被允许的。尽管有括号嵌套，但是括号是一一配对的。

⑤$NaAlO_2+CO_2+H_2O=Al(O+H)_3+Na_2CO_3$，是不被允许的。因为$Al(O+H)_3$会被认为是两种物质，而导致括号不能一一配对。

⑥$NaAlO_2+CO_2+H_2O=Al(OH)_3=Na_2CO_3$，是被允许的。但是不建议，因为程序可能会出现未知错误。

⑦$NaAlO_2+CO_2+H_2O=Al(OH)_3+Na_2CO_3+NO_2$，是被允许的。尽管元素不守恒，但是程序会给出相应的判断。

⑧$NaAlO_2+CO_2+H_2O=Al(OH)_3+Na_2CO_3+*$，是被允许的。但是不建议，因为出现了未知字符"*"，程序可能会出现未知错误。

对于非法的输入，程序可能出现未知错误（"Runtime Error"或"Time Limit Exceeded"或"Unknown Error"）。

但请注意：改变了方程式原意的错误的输入可能不会给出原方程式的正确答案。

三、源代码

整体上，函数名和变量名主要遵循驼峰式命名法，即函数名中的每一个逻辑断点都有一个大写字母来标记：当变量名或函数名是由一个或多个单词连结在一起时，第一个单词以小写字母开始；从第二个单词开始以后的每个单词的首字母都采用大写字母。

代码中还使用到了命名空间来区分不同函数的功能，命名空间变量名和封装的类或结构体的命名规则是：当变量名或函数名是由一个或多个单词连结在一起时，所有单词的首字母都采用大写字母，其余小写。

对于类（class）或结构体（struct）中封装的变量名，也是遵循驼峰式命名法，不过大多数变量名需要在第一个字母前使用下划线标记。

在变量名的使用规则上，使用英文意义。例如，临时变量一般有"cur"、"temp"、"now"等组成，标记用的变量一般有"mark"、"flag"等组成，用"Fir"、"Sec"等标记意义相似的变量。

B.源码

只提供消元部分源码，其它具体见附件。

```
const LD _minN=1E-10;
void gauss(Matrix &_T)
{
    for(int i=1;i<=_T._n;i++)
        _T._num[i][_T._m]=-_T._num[i][_T._m];
    int nowLine=1;
    for(int i=1;i<=_T._n;i++)
    {
        int _current=nowLine;
        for(int j=nowLine+1;j<=_T._n;j++)
            if(fabs(_T._num[j][i])>fabs(_T._num[_current]
[i])&&fabs(fabs(_T._num[j][i])-fabs(_T._num[_current][i]))>=_minN)
                _current=j;
        for(int j=1;j<=_T._m;j++)
            swap(_T._num[nowLine][j],_T._num[_current][j]);
        if(fabs(_T._num[nowLine][i])<=_minN)
            continue;
        LD curFir=_T._num[nowLine][i];
        for(int j=1;j<=_T._m;j++)
            _T._num[nowLine][j]/=curFir;
        for(int j=1;j<=_T._n;j++)
        {
```

```
            if(j==nowLine)
                continue;
            LD curSec=_T._num[j][i]/_T._num[nowLine][i];
            for(int k=i;k<=_T._m;k++)
                _T._num[j][k]-=_T._num[nowLine][k]*curSec;
        }
        nowLine++;
    }
    _T._num[0][0]=nowLine;
}
```

---

1. Gilbert Strang.Introduction to Linear Algebra(fifth edition)[M].Wellesley:Wellesley-Cambridge Press,2016. ↩

2. David C.Lay,等.线性代数及其应用(第5版)[M].刘深泉,等,译.北京:机械工业出版社,2018:4-6. ↩

3. 文传军,等.高斯消元五步骤法[J].常州工学院学报,2012:50-53. ↩

4. Leon S J.Linear Algebra with Applications[M].Macmillan Pub,1986. ↩

5. Cormen T.H,等.算法导论(第3版)[M].殷建平,等译.北京:机械工业出版社,2013.1:478-485. ↩