本题可以采用《编译原理》里面的确定的有限状态机（DFA）解决。构造一个DFA并实现，构造方法可以先写正则表达式，然后转为 DFA，也可以直接写，我就是直接写的，虽然大概率不会是最简结构（具体请参考《编译器原理》图灵出版社），不过不影响解题。DFA 作为确定的有限状态机，比 NFA 更加实用，因为对于每一个状态接收的下一个字符，DFA 能确定唯一一条转换路径，所以使用简单的表驱动的一些方法就可以实现，并且只需要读一遍输入流，比起 NFA 需要回读在速度上会有所提升。

构建出来的状态机如封面图片所示（红色为 终止状态，蓝色为 中间状态）。根据《编译原理》的解释，DFA 从状态 0 接受串 s 作为输入。当s耗尽的时候如果当前状态处于中间状态，则拒绝；如果到达终止状态，则接受。

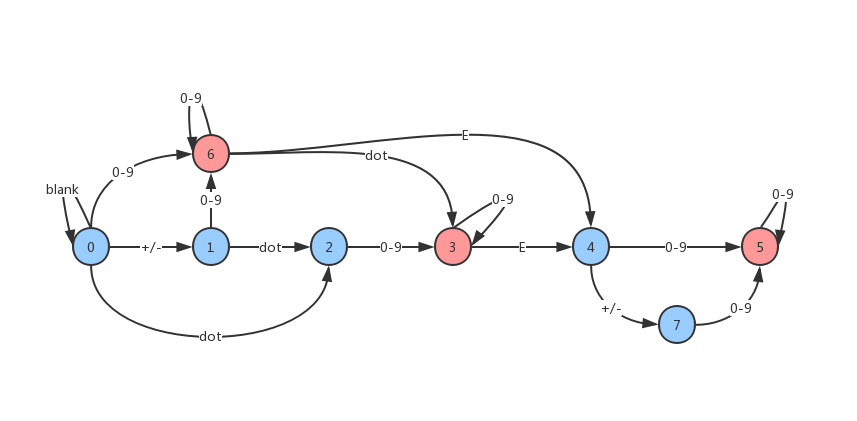
然后，根据 DFA 列出如下的状态跳转表，之后我们就可以采用 表驱动法 进行编程实现了。需要注意的是，这里面多了一个状态 8，是用于处理串后面的若干个多余空格的。所以，所有的终止态都要跟上一个状态 8。其中，有一些状态标识为-1，是表示遇到了一些意外的字符，可以直接停止后续的计算。状态跳转表如下：

0 初始状态 1 符号 2 数字 3 点 4 数字+点

5 数字+E 6 点+数字 7 数字+E+符号 8 数字+E+数字

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| state | Blank | +/- | 0 - 9 | . | E | other |
| 0 | 0 | 1 | 6 | 2 | -1 | -1 |
| 1 | -1 | -1 | 6 | 2 | -1 | -1 |
| 2 | -1 | -1 | 3 | -1 | -1 | -1 |
| 3 | 8 | -1 | 3 | -1 | 4 | -1 |
| 4 | -1 | 7 | 5 | -1 | -1 | -1 |
| 5 | 8 | -1 | 5 | -1 | -1 | -1 |
| 6 | 8 | -1 | 6 | 3 | 4 | -1 |
| 7 | -1 | -1 | 5 | -1 | -1 | -1 |
| 8 | 8 | -1 | -1 | -1 | -1 | -1 |

状态图：



先枚举一下各种合法的输入情况：

1.空格+ 数字 +空格

2.空格+ 点 + 数字 +空格

3.空格+ 符号 + 数字 +　空格

4.空格 + 符号 + 点 +　数字　＋空格

5.空格 + (1, 2, 3, 4） + e +　(1, 2, 3, 4) +空格

0、空格（只有空格，以下八种前面可以有任意数量空格）

1、数（不含小数点的数，可以含正负号）

2、点（可以含正负号）

3、正/负号（只有正负号）

4、点+数（在1基础上含小数点的数）

5、...+e/E（前面合法+e/E）

6、...+e/E+正/负号（前面合法+e/E+正负号）

7、...+e/E+数（此处的数同1，不能有小数点）

8、合法+空格（末尾可以有任意数量空格）