

# 测试报告

## 1 分析错误原因

对于给出的正则表达式  $((+|-)?dd^*)(.dd^*)((E|e)((+|-)?dd^*))?$  提交上去的程序运行出错闪退，原因有两点：

- 完成了实验二的选做内容正闭包（+），导致程序无法正确区分字符'+'与正闭包运算符（+）
- 在对原正则表达式添加显式的连接符号时使用了小数点（.），导致无法正确识别题目中带小数点的正则表达式

## 2 修改正则表达式

根据以上两点错误原因，我将以上正则表达式进行以下两点修改：

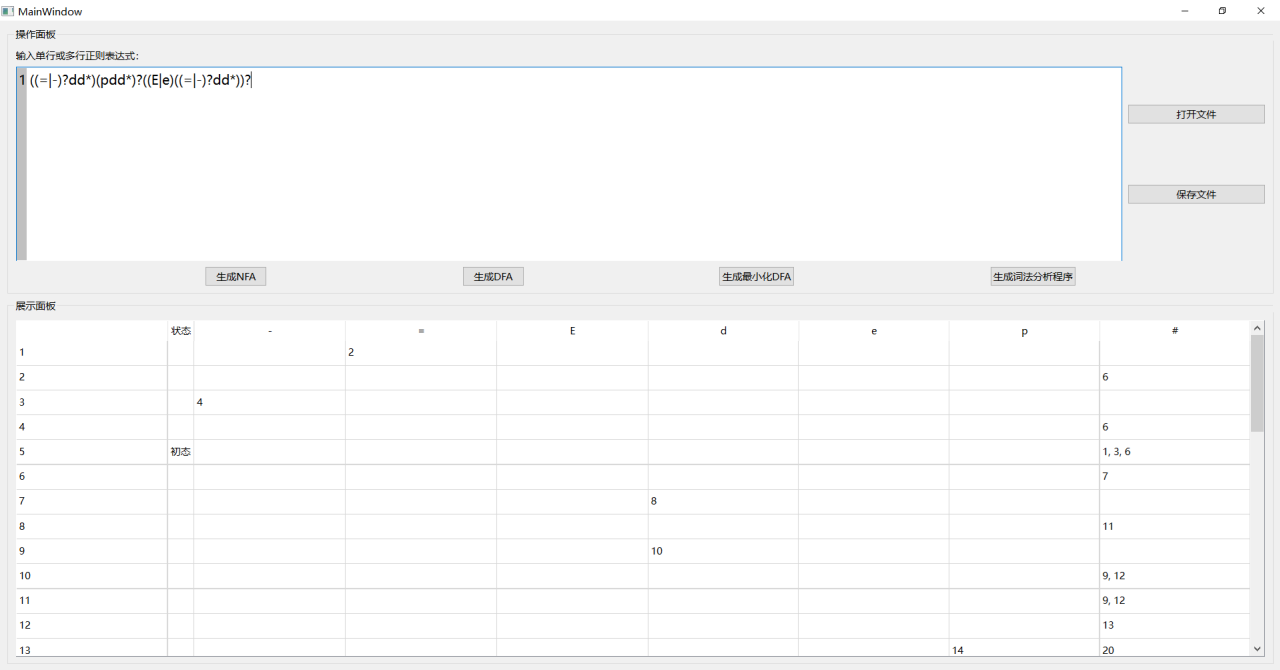
- 将原表达式中的（+）修改为（=）
- 将原表达式中的（.）修改为（p）

修改后的正则表达式为  $((=|-)?dd^*)(pdd^*)((E|e)((=|-)?dd^*))?$ ，修改后程序能正常运行。

## 3 运行结果及分析

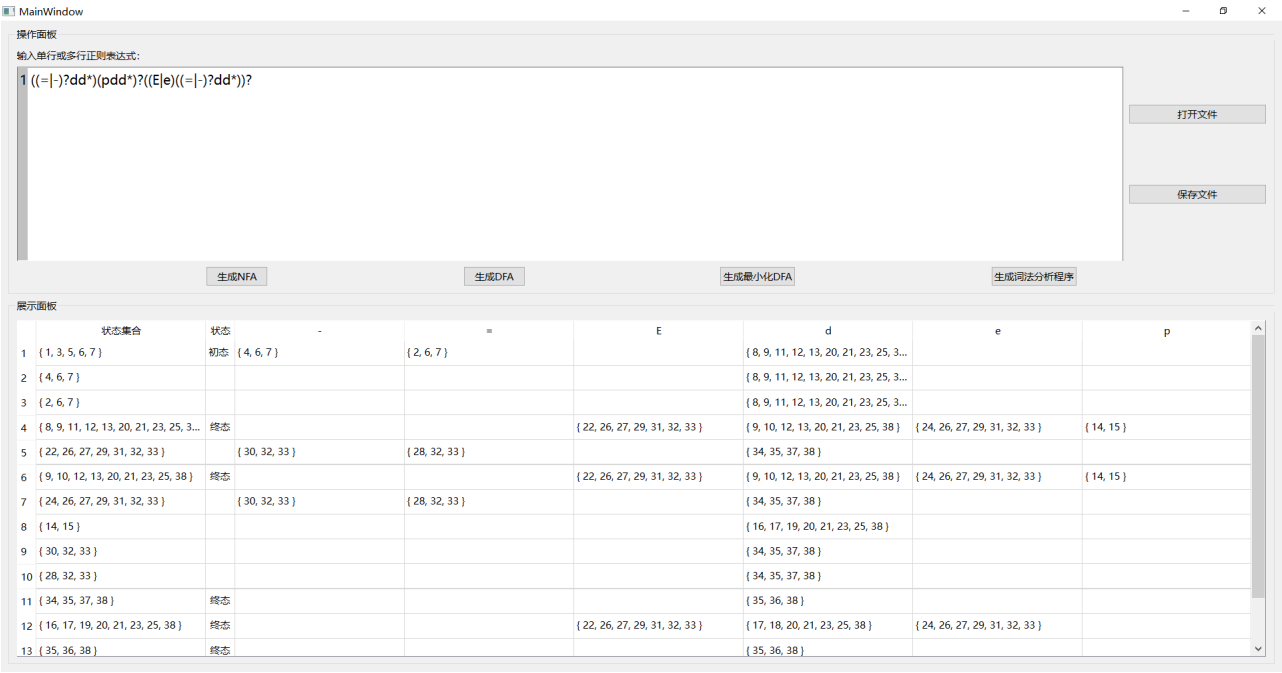
程序运行结果如下图：

### 3.1 生成NFA



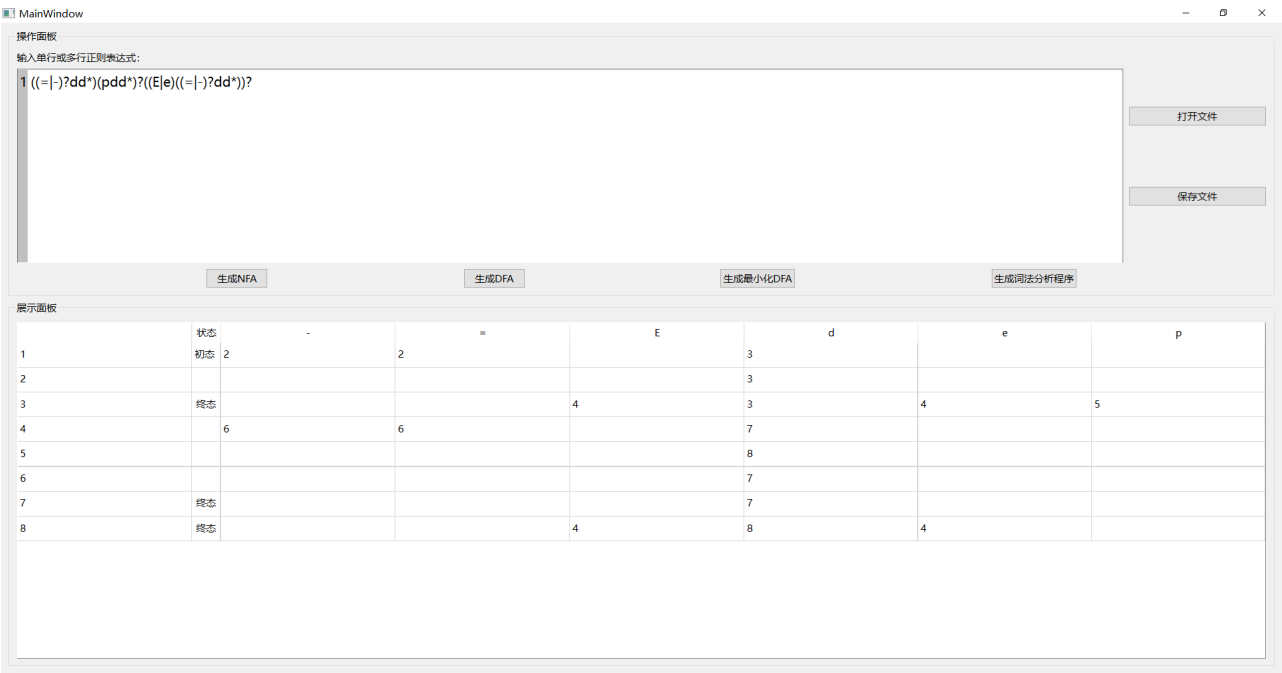
每个字符在**NFA**中需要两个状态表示，选择运算（|）与闭包运算（\*）均需要额外的两个状态来表示，可选运算（?）不需要添加额外的状态。该正则表达式总共有 **13** 个字符，**3** 个选择运算，**3** 个闭包运算，需要的状态数为  $(13 + 3 + 3) \times 2 = 38$  个，而程序生成的**NFA**正好包含 **38** 个状态，可以初步判断程序生成了正确的**NFA**。

### 3.2 生成DFA

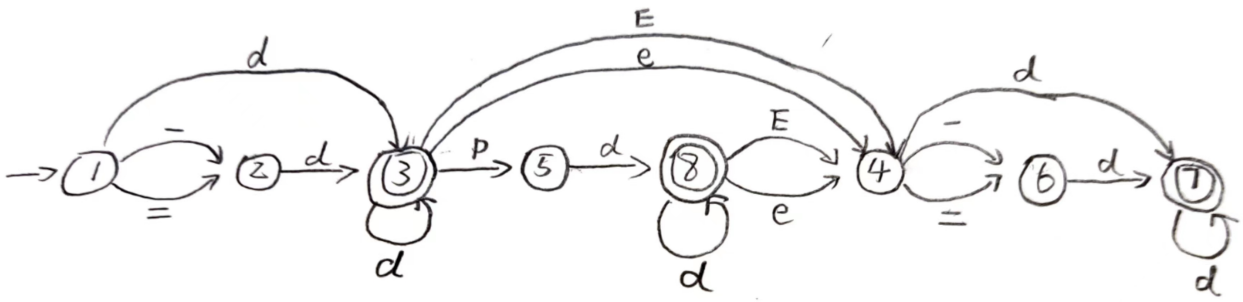


程序能将**NFA**的等价节点用集合表示并输出。

### 3.3 生成最小化DFA



根据生成的最小**DFA**，画出状态转换图如下：



与浮点数的DFA图作对比：

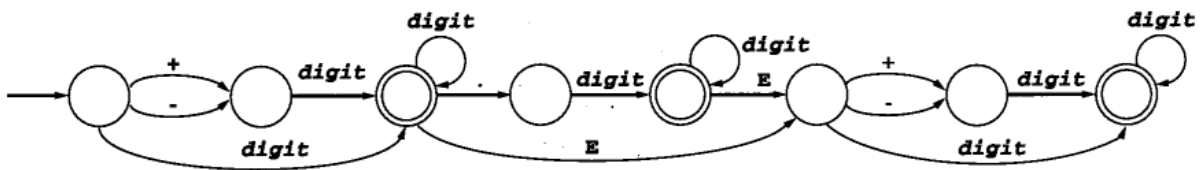


图2-3 浮点数的有穷自动机

可以看出是一致的，因此表明程序生成了正确的最小化DFA。

### 3.4 生成词法分析程序

XLEX
×

```

1 #include<iostream>
2 #include<string>
3 using namespace std;
4 int main() {
5     string s;
6     cin>>s;
7     int state = 1;
8     for (char c : s) {
9         switch (state) {
10             case 1:
11                 switch (c) {
12                     case '-': state = 2; break;
13                     case '=': state = 2; break;
14                     case 'd': state = 3; break;
15                     default: state = 0;
16                 }
17                 break;
18             case 2:
19                 switch (c) {
20                     case 'd': state = 3; break;
21                     default: state = 0;
22                 }
23                 break;
24             case 3:
25                 switch (c) {
26                     case 'E': state = 4; break;
27                     case 'd': state = 3; break;
28                     case 'e': state = 4; break;
29                     case 'p': state = 5; break;
30                     default: state = 0;
31                 }
32                 break;
33             case 4:
34                 switch (c) {

```

保存

将其保存为 **cpp** 文件，测试运行：

The screenshot shows a C++ IDE with a file named `float_regex.cpp`. The code implements a state machine to validate float literals. It starts with `int state = 1;` and processes each character in the input string `s`. The states and transitions are as follows:

- State 1:** Initial state. Transitions: `'-'` to State 2, `'.'` to State 2, `'d'` to State 3, default to State 0.
- State 2:** After a sign or decimal point. Transitions: `'d'` to State 3, default to State 0.
- State 3:** After digits. Transitions: `'E'` to State 4, `'d'` to State 3, `'e'` to State 4, `'p'` to State 5, default to State 0.
- State 4:** After `E` or `e`. Transitions: `'p'` to State 5, default to State 0.
- State 5:** After `p` (exponent part). Transitions: default to State 0.
- State 0:** Invalid state, leads to failure.

The test results panel on the right shows four test cases:

- Testcase 1:** Passed (301ms). Input: `-dd`. Expected Output: 匹配成功. Received Output: 匹配成功.
- Testcase 2:** Passed (47ms). Input: `=-dpd`. Expected Output: 匹配失败. Received Output: 匹配失败.
- Testcase 3:** Passed (48ms). Input: `dddppddEd`. Expected Output: 匹配成功. Received Output: 匹配成功.
- Testcase 4:** Passed (255ms). Input: `=dpde-dd`. Expected Output: 匹配成功. Received Output: 匹配成功.

可以看出程序能正确地对浮点数进行识别。