# 正规式转 NFA 算法代码

## 开发工具

* Visual Stuio Code
* Cmake

**如何编译**

用 Visual Stdio Code 打开项目以后点击左下角的齿轮符号编译。

**Graphical user interface, text, application

Description automatically generated**

## 算法流程

部分参考了这个文章：<https://swtch.com/~rsc/regexp/regexp1.html>

### 正规式解析

首先是正规式解析，在正规式解析的时候，我们需要利用算法，将正规式中缺少的 . 连接符补全。用这样的一个函数补全省略的 . 连接符

Text

Description automatically generated

然后我们需要将正规式转换成后缀表达式，方便解析

Text

Description automatically generated

将正规式转换成后缀式主要就是需要一个栈负责存储运算符，一个字符串变量用来存储临时结果。

主要参考了这篇文章编写的后缀表达式转换函数 <https://www.cxyxiaowu.com/919.html>。

### NFA 构建

接下来就是 NFA 构建。在 NFA 构建的时候，主要用了下面三个结构体来存储 NFA。其中 Node 用来存储 NFA 中的状态节点。Edge 用来存储表示转移函数的 NFA 边。最后用 NFA 结构体来存储 NFA 的开始节点和终结节点信息。

Text

Description automatically generated

然后开始写构建的逻辑。因为现在是后缀表达式，只需要用一个栈来存储运算的中间结果，就能构建完成整个 NFA。具体的后缀表达式解析过程，在 <https://www.cxyxiaowu.com/919.html> 中也有提及。构建的时候，总共需要对 . \* | 这三种运算符分别进行处理。

* 其中 . 就是将栈中的临时结果取出两个，然后在将这两个小 NFA 用一条空边连接起来。
* \* 就是取出栈中的一个临时 NFA，然后在这个临时 NFA 中增加两个节点和适当的边构建闭包
* | 就是从栈中取出两个临时 NFA，然后在新的 NFA 中增加两条路线分别经过这两个 NFA。

最后的生成结果如下：

A picture containing table

Description automatically generated Shape, arrow

Description automatically generated