设计说明

采用了栈的设计，对输入符号串进行处理.

**数据结构说明**

struct edge

{

char weight;

char next;

};

string option,expression, suffix, infix, postion;

map<char, int> precedence;

map<char, vector<edge>> nfa;

vector<string> tfunction;

// 具体解释如下：

// edge：NFA图中的边

// option：输入的符号集

// expression：输入的正规式

// infix：增加连接符的正规式

// suffix：转换成的后缀表达式

// postion：状态集

// precedence：规定运算符优先级

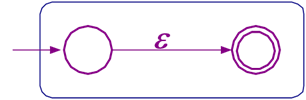
// nfa：生成的NFA（使用map生成了一个char与edge向量的映射，char代表NFA中的顶点，也就是生成了一个顶点与从顶点出发边的集合的映射）

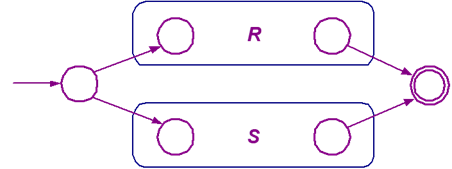
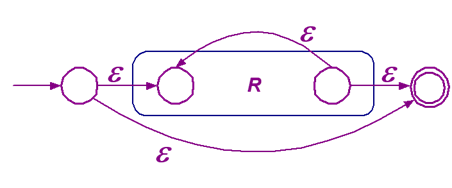
// tfunction：转移函数

**思路**

使用Thompson构造法，输入为字母表Σ上的正规式，输出为相等的NFA。具体方法为：

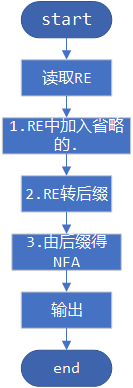
首先将构成r的各个元素分解，对于每一个元素，按下述规则1和规则2生成NFA。（注：如果r中记号a出现了多次，那么对于a的每次出现都需要生成一个单独的NFA）之后依照正规表达式r的文法规则，将生成的NFA按照下述规则3组合在一起。

1. **规则1** 对于空记号ε，生成下面的NFA。  
   
2. **规则2** 对于Σ的字母表中的元素a，生成下面的NFA。  
   
3. **规则3** 令正规表达式R和S的NFA分别为N(R)和N(S)。

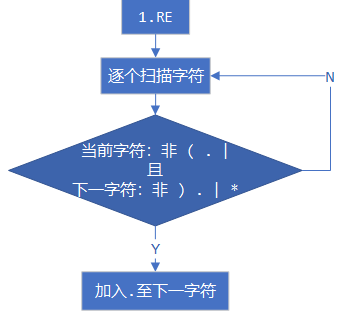
* 对于R|S，按照以下的方式生成NFA N(R|S)。  
  
* 对于R.S，按照以下的方式生成NFA N(RS)。  
  
* 对于R\*，按照以下的方式生成NFA N(R\*)。  
  

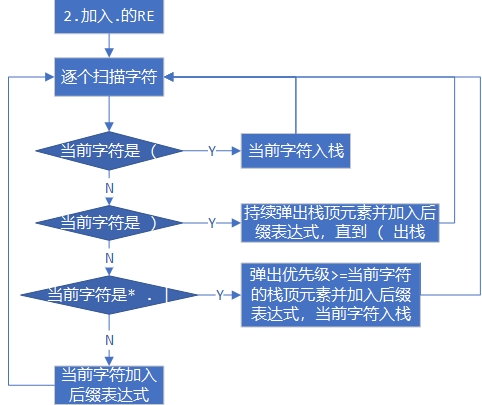
**流程图**

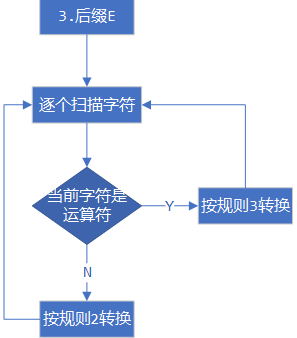
* 总体流程图



* 在正规式中加入省略的连接符



* 将加入连接符的正规式转为后缀表达式
* 将后缀表达式转换成NFA



**执行结果**

