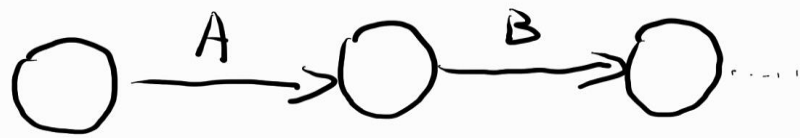


词法分析——正则表达式解析

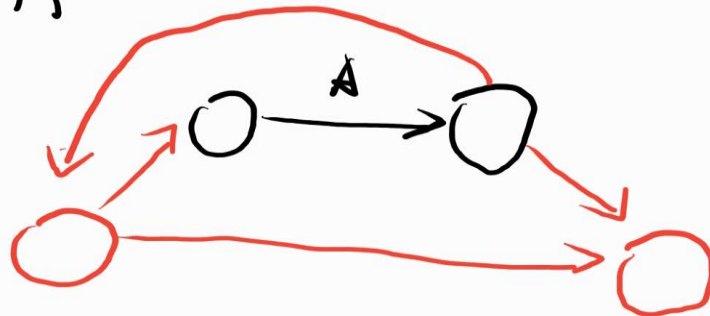
为了解析正则表达式，我们这里需要两个栈，一个栈负责记录操作符，另一个负责记录“节点块”。在这里操作符包括 $(,), *, +, |$ 。而所谓“节点块”则表示整个状态的一部分。

在读取字符串的过程中，我们需要构建状态机的图结构有如下几种。

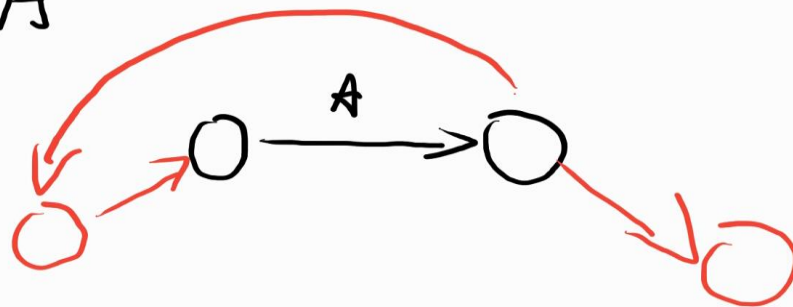
$AB\dots$

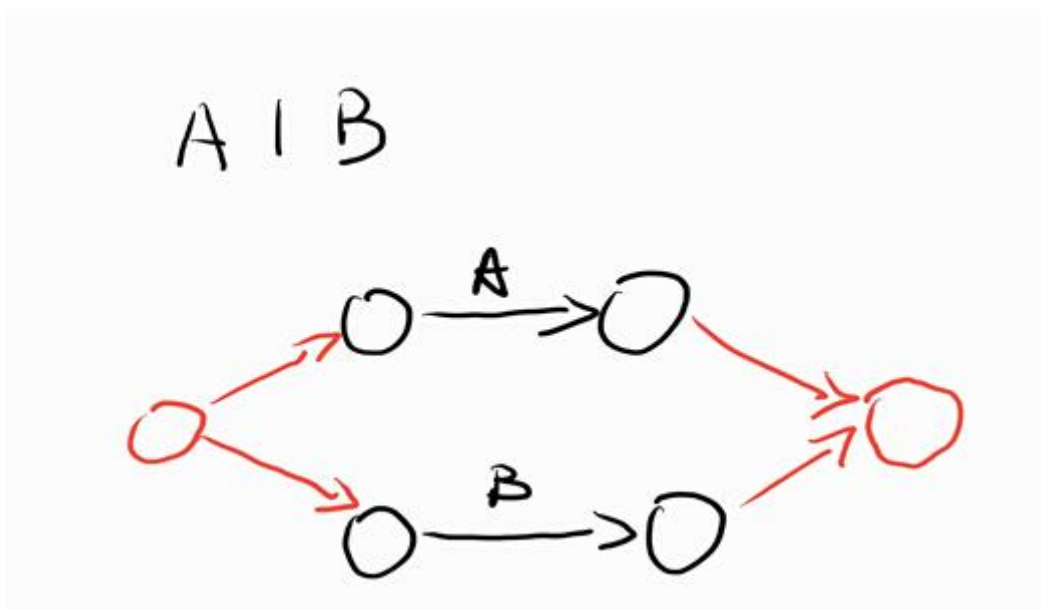


A^*



A^+





在图结构中，节点代表是状态，而节点间的边是单向的，并且可能会附带信息，如果附带有信息则仅在该条件下才会发生状态转换，而如果边上没有任何条件则代表可以在任何条件下发生状态转换。

从上图我们可以看到，对于普通的字符判断（ AB ）只需要建立一个初始节点，然后按字符顺序依次建立线性的状态转换关系即可。

对于 $*$ 运算符和 $+$ 运算符，我们则需要额外建立两个节点，对其管辖范围的节点块的首节点和尾节点按图中红色部分建立拓扑关系即可。例如 $(A|B)^*$ 这个表达式，我们需要先建立 $A|B$ 的图结构（如上图），然后取其首节点和尾节点（图中红色的节点）替换掉 A^* 中黑色的部分即可。

而对于 $|$ 运算符而言，我们同样需要建立两个额外的节点，然后建立如图所示拓扑关系。如果左右两边的结构比较复杂，处理的方法同 $*$ 和 $+$ 运算符，只需要替换图中黑色的部分就可以了。

了解了这些之后，我们便可以开始解析字符串了。从左到右一个字符一个字符的读取字符串。

如果读入的是（，则将它压入到操作符栈中，继续读取下一个字符。

如果读取到的是字符（非上文提到的操作符），则继续向后读取直至连续的最后一个字符（非上文提到的操作符），然后建立相应的图结构（按照上面的图），然后将它节点块信息压入到另一个栈中，继续读取下一个字符。

如果读取到的是 $*$ 操作符或者 $+$ 操作符，则将其压入到操作符栈中，继续读取下一个字符。

如果读取到)，则弹出操作符栈的栈顶元素，如果该元素不是(，则从存储节点块信息的栈中取出并弹出栈顶元素，然后按照上面的图建立相应的图结构，并把新的节点块信息压入到这个栈中。重复上述操作直到栈顶元素为(，这时什么都不做，弹出(后继续都下一个字符。

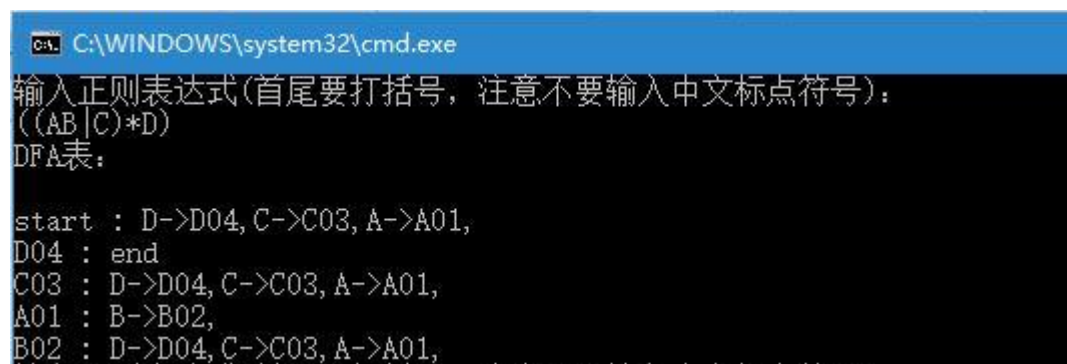
当字符串读取完毕时，操作符栈应该为空（只要保证正则表达式的首尾有一对括号即可）。

最后只需把节点块栈中的节点块从栈底到栈顶按顺序连接起来就可以了，这样我们就得到了一个 DFA。

然后我们需要把图中的无条件转换的边去掉，方法是：依次检索所有通过条件转换而来的节点和起点，首先获得**这个节点**在无条件下能够到达状态的结合，将这个集合中的某个节点能通过条件状态转换能到达另一个节点，那么将这条边复制到**这个节点**（同前文加粗的节点）上。最后删掉除了有通过条件转换而来的节点和起点以外的所有节点和无条件的空边即可。

然后将上述的图结构转换为表结构，再使用老师上课所讲的算法（使用一个队列和一个集合即可实现）即可将 DFA 所对应的表转换 NFA 所对应的表结构。

程序运行效果如下（运行环境 .NET 4.5 VS2013）：



```
C:\WINDOWS\system32\cmd.exe
输入正则表达式(首尾要打括号，注意不要输入中文标点符号):
((AB|C)*D)
DFA表:

start : D->D04, C->C03, A->A01,
D04 : end
C03 : D->D04, C->C03, A->A01,
A01 : B->B02,
B02 : D->D04, C->C03, A->A01,
```