实验报告

苟芳菲 2021011837 软件13

1. 编程语言：c++
2. 编程环境：
    1) 系统：Windows 11
    2) OS版本：22621.1265
    3) 处理器：AMD Ryzen 5 5600U with Radeon Graphics 2.30 GHz
    4) 系统类型：x64
    5) 编译器：MSVC 19.33.31629.0
3. 主要实现思路
    1) Lab1 判断NFA能否接受字符串
        a) 深度优先的思路，走到非终态且不能转移的情况下再返回重新走
        b) 通过调整每个state对应的rule的顺序来实现贪婪和非贪婪的选择
    2) Lab2 构造NFA
        a) 学习regex文法，理解符号的对应关系
        b) 深度优先的思路，取出当前符号并做转移处理后递归构造“小自动机”
        c) 对特殊符号做特殊的转移处理，如“*”“+”“？”
        d) 对字符正常构造状态转移
        e) 由于要求字符匹配不一定从开头开始，因此在match函数中不断对取出第一个字符后的新字符串输入NFA进行匹配操作直到有返回path或字符串输完
    3) Lab3 补充完整文法
        a) 捕获分组：记录要捕获的起始state，在Lab1中给出的path.state里找到对应的起始state，再利用path.consume的定义输出分组
        b) anchor字符：在构造NFA和字符串匹配时都做特殊处理，将其视为有条件的epsilon转移
        c) flag：m情况下将判断是否为开头和结尾的条件进行扩充，s情况下特别增添\r,\n即可
        d) matchALL：由于要求全部输出且不覆盖，因此在有输出时不要直接return而是最后再return，并且到下一次匹配shift的长度为上一次匹配的字符串

长度

e) 区间限定符：将其理解为类似()*的形式，同样地，这类转移有次数限制

4. 重难点问题及解决方法

1) 问题：Lab2 中非贪婪匹配时是要把 epsilon 转移放在 rule 的最后，但是由于构造自动机时是由小到大递归构造，所以只是把这条 rule 插在了当前 rules 的后面

解决：和贪婪匹配一样将这条 rule 插到前边，在 match 里再将最后的完整 rules 逆序

2) 问题：Lab2 中处理取反的情况时，逻辑上是所有出现的路径都不能走，但是如果直接在代码里添加，会出现只会不走第一条 rule 的情况

解决：重新处理有取反的情况，创建一个二维数组，dst 相同的在同一行，对同一行 rule，只要有一个取反（reverse = true）的情况，要判断整行是否能转移，只要有一个不能转移就置 flag 为 false，只有 flag 为 true 时才真正能转移

3) 问题：Lab3 中处理/b 时，是将其作为特殊的 epsilon 转移看待，但是由于 Lab1 中输出 path 时我采用了最后一并 push_back 的做法，所以当\b 在正则表达式的开头时，最终会输出匹配上\b 的转移的那个字符，即把单词边界的前一个字符也输出

解决：用一个 vector 专门存储是否是单词边界的特殊转移，如果是的话，最后构造 consume 数组的时候就不讲对应的字符插入数组，如果不是，继续按照之前的方法插入消耗字符