XLEX词法生成器

一、设计要求

设计一个应用软件，实现正则表达式→NFA→DFA →DFA最小化 →词法分析程序，其中正则表达式支持单个字符，运算符号支持"|", "()", "*", "+", "?", "[]", 连接运算。要求提供编辑界面，用于输入正则表达式，提供窗口分别用于观察NFA, DFA, miniDFA和词法分析程序，其中词法分析程序可以保存为.cpp。

二、设计思路

1. 准备工作

   用邻接表的方法建立三张图分别用来存储NFA，DFA，miniDFA。

2. regEx→NFA

   首先根据运算符号的优先级将正则表达式转换为后序表达式，然后定义多个运算符号函数用于实现将结点和边插入到NFA中。

3. NFA→DFA

   首先定义ε闭包函数(即找出起始点经过 达到的所有点)，然后从NFA的起始点出发利用 闭包函数求起始点的 闭包，以NFA起始点的 闭包作为DFA的起始点. 再遍历regEx中出现的所有字符，判断DFA 起始点中是否有点存在边经过字符，分别统计经过不同字符的终点，将他们作为新的顶点，并分别插入到DFA图中，接着再把这些新的顶点作为起始点继续执行前面这个过程(找起始点的 闭包，遍历字符集，找新的顶点)。

4. 最小化DFA

   维护一个map统计DFA图中各个顶点经过各个字符后的状态是否相同(到达同样的结点)，如果DFA图中两个顶点的状态相同，则把它们视为同一个顶点，利用这个办法把DFA图中的各个顶点划分为不同的分组，然后根据这个分组再进行图的顶点插入和边插入。

三、一些函数的声明

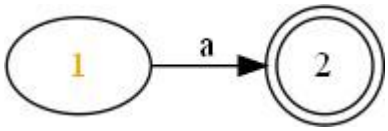
```
//  NFA和DFA中基本运算的函数声明
void basic(QChar c);     //基本正则表达式
void apposition();       //并置 &
void selection();        //选择 |
void closure();          //闭包
void optional();         //可选
void positiveClosure(); //正闭包
void epsilonClosure(int start, QSet<int> &set); //epsilon闭包
```

```
//将一个set转换成"{1,2,3}"这种形式的字符串，   且判断set是否包含了终止点
QString setToQStringAndJudgeWhetherEnd(QSet<int> &set, bool &isEnd);
//判断基本运算的优先级
int getPriority(QChar c);
```

四、具体思路

regEx→NFA

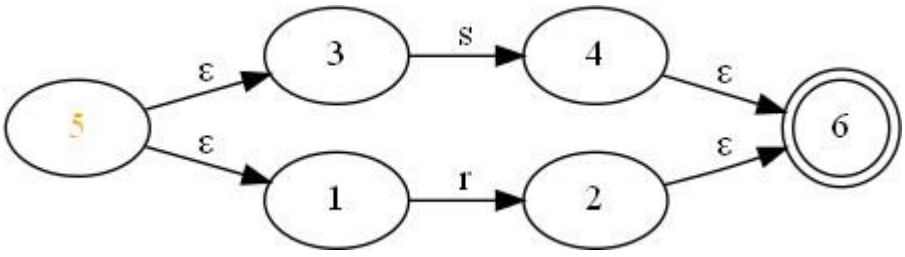
1. 将regEx中的连接运算改为"&",比如"ab"变成"a&b"； 将"[]"转换成"(|)"，比如"[a-c]"变成"(a|b|c)"
2. 利用栈将变换后的regEx变成后序表达式，详细的思想是遍历表达式，如果遇到字符不是运算符就直接插入到结果字符串中；如果遇到的字符是运算符且不是"("或者")"，如果栈为空则将运算符push到栈中，如果栈不为空，则将栈中比该运算符优先级高的运算符pop出("("除外)，且插入到结果字符串中；如果遇到"("则直接push到栈中；如果遇到")"则将栈中元素pop出且插入到结果字符串中，直到遇到"("；最后如果栈不为空，则把栈中的元素全部pop出且插入到结果字符串中
3. 利用stack< int >存储NFA当前的起始结点序号和结束结点序号
   - 遇到非运算符，如'a'，其NFA图为



所以应该向NFA插入两个结点，再创建一条边由前一个结点指向后一个权值为a，具体代码为

```
void utils::basic(QChar c)
{
    //插入两个结点， 第一个为起始， 第二个为终止， 并获取它们在NFA中的编号
    //分别为nfaGraph.getVertexSize(), nfaGraph.getVertexSize() + 1
    int vertexSize = nfaGraph.getVertexSize();
    nfaGraph.insertVertex();
    nfaGraph.insertVertex();
    //用stack存储序号
    stack.push(vertexSize);
    stack.push(vertexSize + 1);
    //插入边
    nfaGraph.insertEdge(c, vertexSize, vertexSize + 1);
}
```

   - 遇到运算符'|'，如"rs|"(后序表达式)，其NFA为

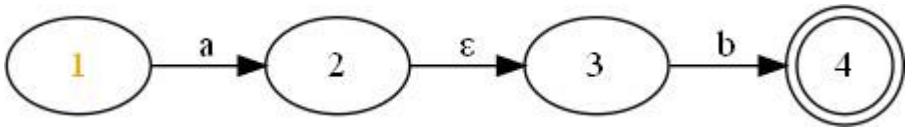


先从stack中分别获取s和r的开始结点序号和结束结点序号，然后向NFA中插入两个新结点作为新的开始结点和结束结点，再根据上图所示插入4条边，具体的代码为

```
void utils::selection()
{
    //分别获取s和r的开始结点序号和结束结点序号
```

```
    int secondEnd = stack.pop();
    int secondStart = stack.pop();
    int firstEnd = stack.pop();
    int firstStart = stack.pop();
    //插入两个新结点
    int vertexSize = nfaGraph.getVertexSize();
    nfaGraph.insertVertex();
    nfaGraph.insertVertex();
    //插入4条边
    nfaGraph.insertEdge(epsilon, vertexSize, firstStart);
    nfaGraph.insertEdge(epsilon, vertexSize, secondStart);
    nfaGraph.insertEdge(epsilon, firstEnd, vertexSize + 1);
    nfaGraph.insertEdge(epsilon, secondEnd, vertexSize + 1);
    //在stack中存储新结点的序号
    stack.push(vertexSize);
    stack.push(vertexSize + 1);
}
```

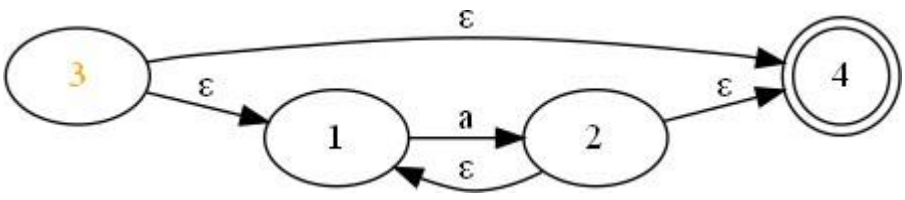
- 遇到运算符'&', 如"ab&"(后序表达式), 其NFA为



先从stack中分别获取a和b的开始结点序号和结束结点序号, 然后再a的结束结点和b的开始结点间插入一条边, 再以a的开始结点和b的结束结点作为新的开始结点和结束结点

```
void utils::apposition()
{
    int rightEnd = stack.pop();
    int rightStart = stack.pop();
    int leftEnd = stack.pop();
    int leftStart = stack.pop();
    nfaGraph.insertEdge(epsilon, leftEnd, rightStart);
    stack.push(leftStart);
    stack.push(rightEnd);
}
```

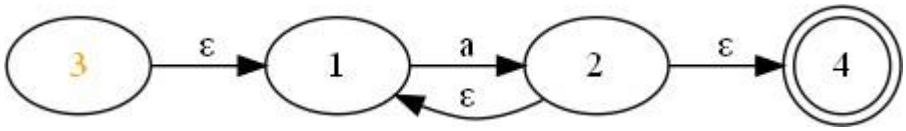
- 遇到运算符'*', 如"a|*"(后序表达式), 其NFA为



获取a的开始结点序号和结束结点序号, 然后向NFA图中插入两个新结点, 然后按如图所示插入4条边, 具体的代码为

```
void utils::closure()
{
    int end = stack.pop();
    int start = stack.pop();
    int vertexSize = nfaGraph.getVertexSize();
    nfaGraph.insertVertex();
    nfaGraph.insertVertex();
    nfaGraph.insertEdge(epsilon, end, start);
    nfaGraph.insertEdge(epsilon, end, vertexSize + 1);
    nfaGraph.insertEdge(epsilon, vertexSize, start);
    nfaGraph.insertEdge(epsilon, vertexSize, vertexSize + 1);
    stack.push(vertexSize);
    stack.push(vertexSize + 1);
}
```

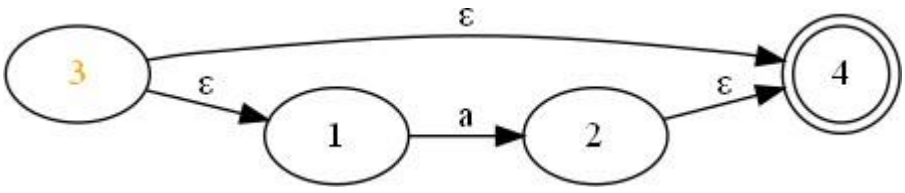
- 遇到运算符'+', 如"a+", 其NFA为



获取a的开始结点序号和结束结点序号, 然后向NFA图中插入两个新结点, 然后按如图所示插入3条边, 具体的代码为

```
void utils::positiveClosure()
{
    int end = stack.pop();
    int start = stack.pop();
    int vertexSize = nfaGraph.getVertexSize();
    nfaGraph.insertVertex();
    nfaGraph.insertVertex();
    nfaGraph.insertEdge(epsilon, end, start);
    nfaGraph.insertEdge(epsilon, end, vertexSize + 1);
    nfaGraph.insertEdge(epsilon, vertexSize, start);
    stack.push(vertexSize);
    stack.push(vertexSize + 1);
}
```

- 遇到运算符'?', 如"a?", 其NFA为



获取a的开始结点序号和结束结点序号, 然后向NFA图中插入两个新结点, 然后按如图所示插入3条边, 具体的代码为

```
void utils::optional()
{
    int end = stack.pop();
    int start = stack.pop();
    int vertexSize = nfaGraph.getVertexSize();
    nfaGraph.insertVertex();
    nfaGraph.insertVertex();
    nfaGraph.insertEdge(epsilon, vertexSize, start);
    nfaGraph.insertEdge(epsilon, vertexSize, vertexSize + 1);
    nfaGraph.insertEdge(epsilon, end, vertexSize + 1);
    stack.push(vertexSize);
    stack.push(vertexSize + 1);
}
```

- 最后将stack中的元素pop出，stack的栈顶元素为NFA的结束顶点，栈顶下面的元素为开始结点。整个过程最关键的是用stack存储NFA的结点序号。

NFA→DFA

1. 从NFA的开始结点出发，找开始结点的ϵ闭包，存入集合中，并将这个集合存入栈中。
2. 弹出栈顶元素，获取当前要遍历的集合，遍历出现的字符，判断集合中的元素是否有边经过该字符，如果有，则把边的终点元素的ϵ闭包插入到临时集合中，最后把集合push进栈中，在DFA中插入一条边权值为当前遍历的字符，start，end分别为当前集合和临时集合，且如果DFA中不存在结点值为该集合，则向DFA中插入值为该集合的结点.
   比如当前要遍历的集合为{1，2，3}，字符集合为{a，b}，遍历a时，2有一条边权值为a，边的终点为4，4的ϵ闭包为{4,5}，把{4，5}push进stack中，然后向图中插入结点值为{4，5}，且插入一条边权值a，起始点为{1，2，3}，终止点位{4，5}；再遍历b，继续执行上述操作。
3. 重复操作2，直到栈为空，注意需要维护另一个set去判断当前要遍历的集合是否遍历过了，如果遍历过了则跳过此次遍历，如果set中不存在该集合，则把集合插入到set中，且执行这次遍历。
4. 为了便于存入图中，实现了一个函数**QString setToQStringAndJudgeWhetherEnd(QSet< int > &set, bool &isEnd);** 将set转换成字符串，且判断该set中是否存在NFA的结束结点。

DFA→miniDFA

1. 利用map统计DFA中各个结点经过是否经过字符集合中的元素，并判断经过后是否到达DFA的终点，把这些结果作为map的value，关键代码为
   利用**QVector<QVector<int>> map**记录DFA中各个结点经过哪些边以及到达对应的点，关键代码为
```
QSet<int> end;
typedef QPair<QChar, int> p;
QVector<QVector<p>> map;
map.push_back(QVector<p>());
QSet<int> del;
Vertex * dfaNodeTable = dfaGraph.getNodeTable();
int vertexSize = dfaGraph.getVertexSize();

for (int t = 1; t < vertexSize; t++) {
    Edge *edge = dfaNodeTable[t].adj;
    if (dfaNodeTable[t].isEnd)
        end.insert(t);
    map.push_back(QVector<p>());
    while (edge != nullptr) {
        map[t].push_back(p(edge->character, edge->dest));
        edge = edge->next;
    }
    std::sort(map[t].begin(), map[t].end(), [](p a, p b)->bool{return a.first < b.first;});
}
```

2. 根据map中的信息进行分组，如果结点的状态相同(都是结束结点或者都不是)且经过相同的字符并且经过相同字符时终点的状态相同，则把结点作为同一组，最后得到集合**QVector<QSet< QString >> group**，根据map中的信息进行分组，如果经过边的权值相同且结点相同，则把他们分为一组，比如:

   1结点经过a到达3，经过b到达4，2结点经过a到达3，经过b到达4，就把1，2看成一组，

   最后得到集合**QVector<QVector < int >> group**，每个**QVector<int>**存放同一组的结点.

3. 继续遍历**map, 每一个组只保留QVector<int>**的第一个元素, 比如1,2 是一组只保留1, 3, 4是一组只保留3, 且如果1有一条边到4, 则把这条边改为1到3.
4. 重复执行过程2, 直到group中的每一个**QVector<int>**的size都为1, 然后在miniDFA插入对应的边和结点。

miniDFA→C++代码

1. 定义函数bool isRight(string s); 判断s是否满足正则表达式
2. 利用while –switch –case的结果生成代码, 获取miniDFA的vertexSize, 以1 ~ vertexSize -1作为while的判断条件
   while ((state == 1 || state == 2|| ... state == vertexSize − 1) && t < s.length())
3. 遍历miniDFA, 以miniDFA结点的编号作为case state:, 以遍历当前结点所有的edge, 以if (s[t] == edge→character) 判断, 如果符合则state = edge→dst

```
//遍历边
Edge *edge = dfaNodeTable[t].adj;
    while (edge != nullptr) {
        //获取结束结点的值
        int dst = edge->dest;
        QString dstData = dfaNodeTable[dst].data;
        bool isEnd2 = dfaNodeTable[dst].isEnd;
        //所在group的下标
        int endIndex;
        for (endIndex = 0; endIndex < groupSize;endIndex++)
            {QSet<QString> set = group[endIndex];
            if (set.find(dstData) != set.end())
                break;
        }
    //插入边，结点
    int graphEndIndex =
miniDfaGraph.findIndexByData(QString::number(endIndex  +  1));
    if (graphEndIndex == -1) {
        graphEndIndex = miniDfaGraph.getVertexSize();
        miniDfaGraph.insertVertex(QString::number(endIndex + 1), isEnd2);
    }
    miniDfaGraph.insertEdge(edge->character,  graphStartIndex,
graphEndIndex);
    edge = edge->next;
}
```

4. 统计miniDFA所有的结束结点，如果最后state==结束结点且t == s.length()则return true，否则return false;

使用GraphViz画图

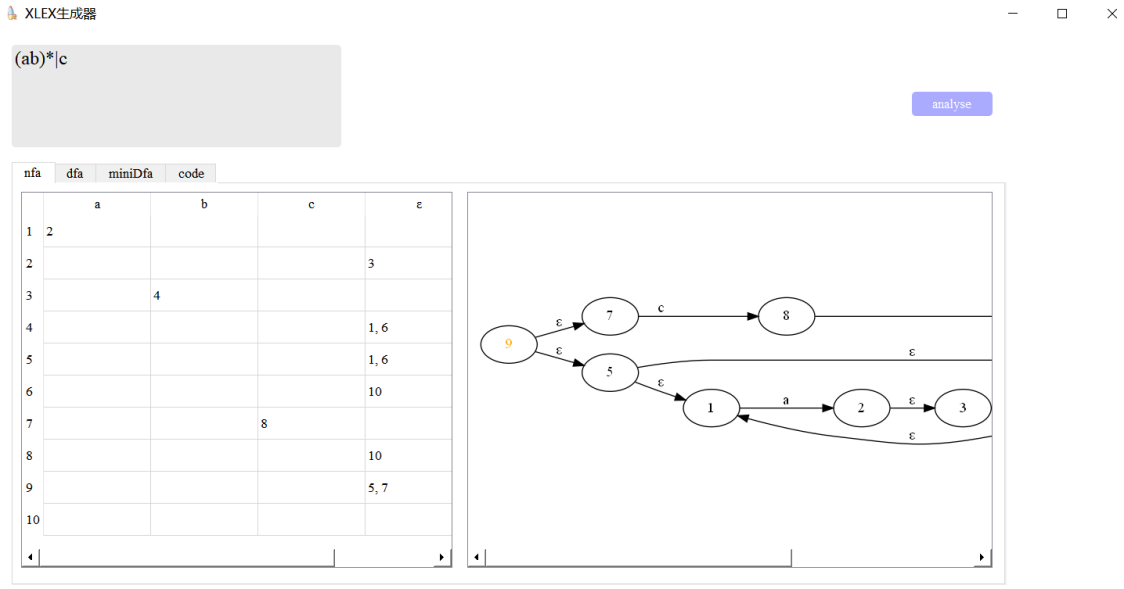
得到NFA，DFA，miniDFA后，先根据GraphViz画图的格式要求，遍历graph生成对应格式的txt文件，再调用GraphViz所用的dot.exe生成对应的png，如果使用源程序直接编译需要在编译结果的debug目录放置Graphviz.

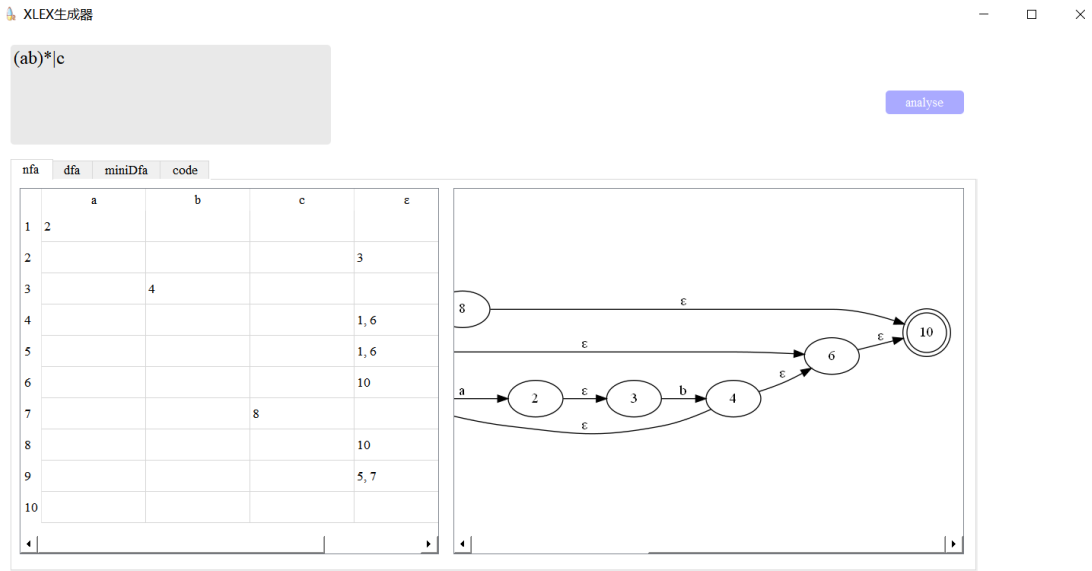
五、实现的语言和工具

1. 实现语言：C++
2. 实现平台：Qt Creator
3. 画图工具：GraphViz

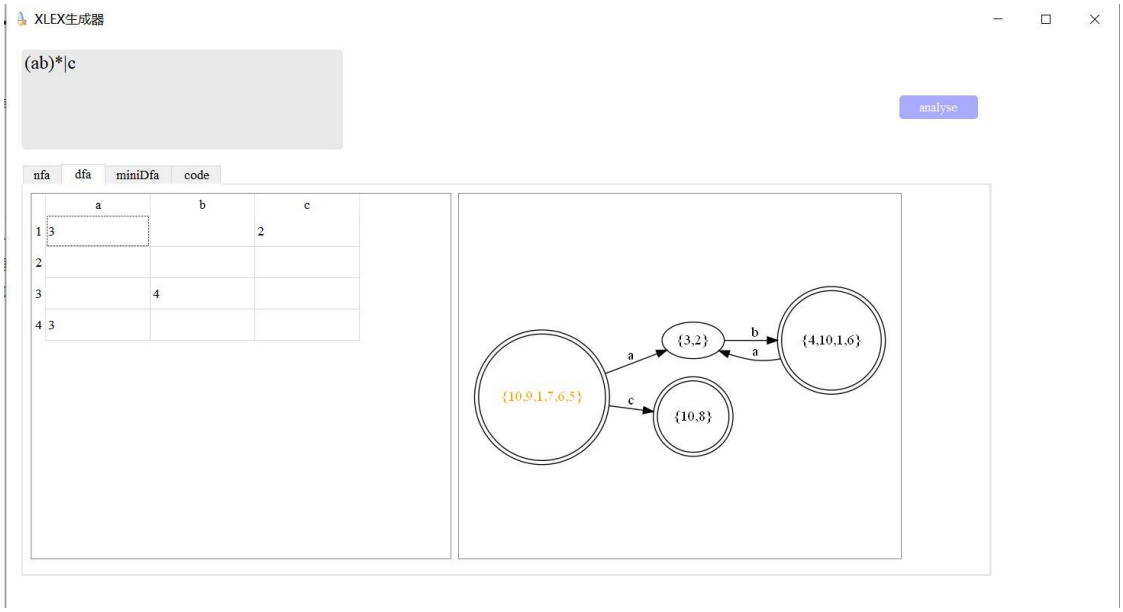
六、操作流程

1. 输入**(ab)*|c**，点击analyse按钮，开始分析
2. NFA：(橙色结点表示开始结点，双圆结点表示结束结点)

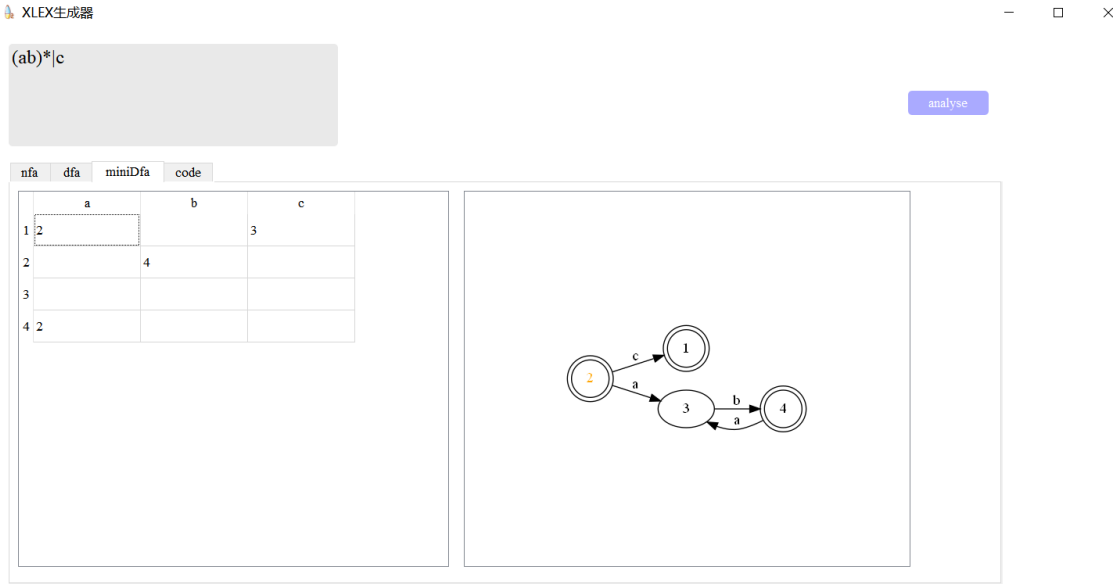




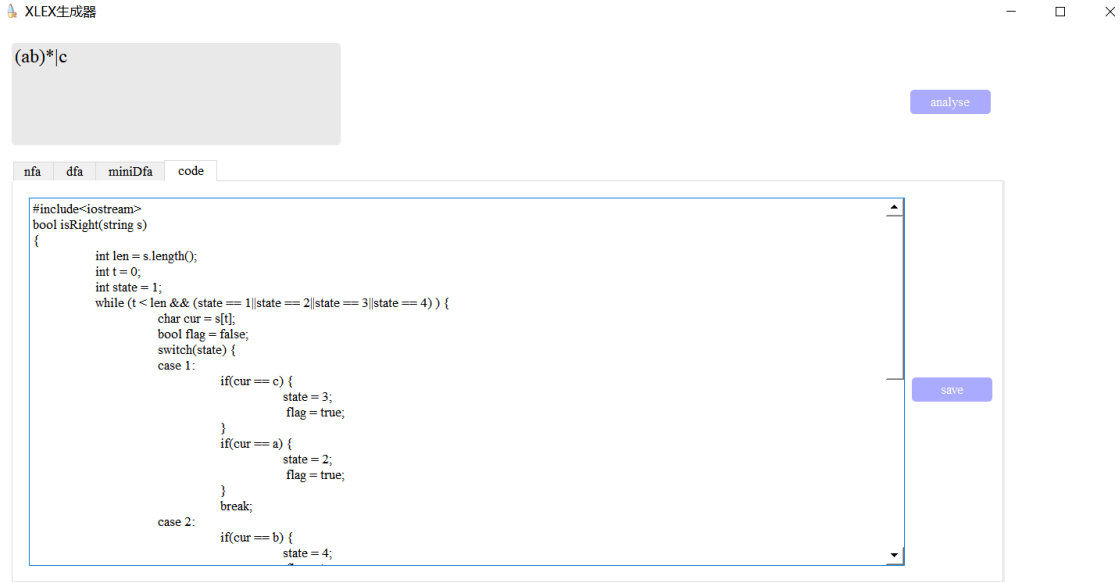
3. DFA: (橙色结点表示开始结点，双圆结点表示结束结点)



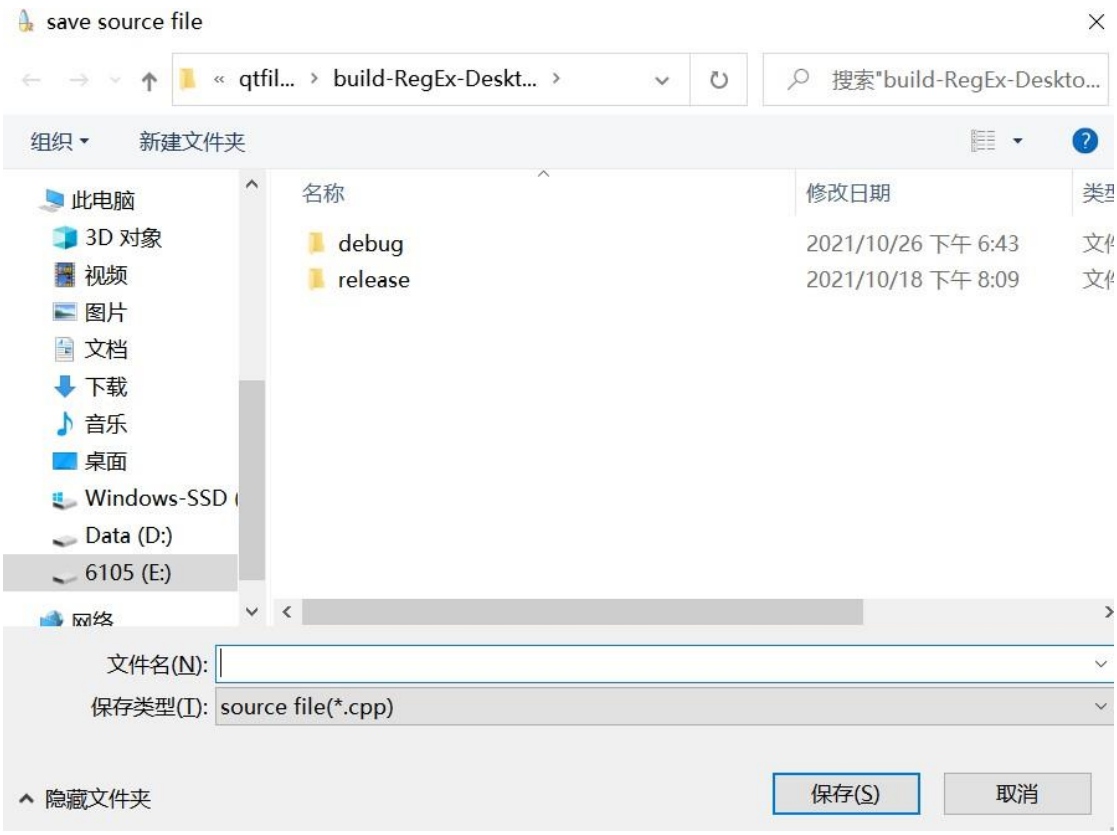
4. miniDFA: (橙色结点表示开始结点，双圆结点表示结束结点)



5. C++代码



点击save按钮，保存文件


save source file
« qtfil... › build-RegEx-Deskt... ›
搜索"build-RegEx-Deskto...
组织
新建文件夹
此电脑
3D 对象
视频
图片
文档
下载
音乐
桌面
Windows-SSD
Data (D:)
6105 (E:)
网络
名称
修改日期
debug
2021/10/26 下午 6:43
release
2021/10/18 下午 8:09
文件名(N):
保存类型(T):
source file(*.cpp)
隐藏文件夹
保存(S)
取消