1. 实验项目分析与设计

1.1 正则表达式到NFA

对于一个给定的正则表达式，可以通过epsilon连接轻松建立NFA。因为正则表达式需要考虑优先级问题，可考虑使用表达式树或后缀表达式来处理优先级。又考虑epsilon连接各个子表达式，可直接在表达式树上进行操作，因此采用表达式树来建立NFA。这一部分使用了比较暴力的时间复杂度为O(n^2)的写法，可用二分定位字符将复杂度降至O(nlogn)，但考虑到整个算法的主要时间消耗不在这个步骤（后面的算法时间复杂度更高），且正则表达式一般不会达到几千以上个字符，又为了方便程序实现和减少bug的出现，就没有实现这一个优化。

1.2 NFA到DFA

将NFA的所有结点该用顺序存储方式，再根据教材上的算法进行转换。

1.3 DFA最小化

根据教材算法进行转换。

2. 具体实现

2.1 正则表达式到NFA

\* 以下“子串”表示正则表达式的一个子字符串。

\* 使用递归的方式建立表达式树。

\* 每层递归返回一个pair<node\*,node\*>，表示子树对应的子图的起点/终点。

\* 将字符串中被括号括起来的部分视为一个整体，在当前子串不考虑括号内的任何字符，单个外部的字符也被视为一个整体。

\* 考虑对于表达树上一棵子树以及对应的子串sub，如果sub中存在’|’字符，则说明sub可以按照’|’来划分，所以就按照’|’划分后向下递归，并用epsilon把多个子节点进行连接。

\* 若不存在’|’字符，则把sub的第一个整体与剩下的部分进行epsilon连接。注意如果第一个整体后跟着\*字符，则要尾首多连一条边。

\* 递归的边界就是sub中只剩一个字符。

代码中实现这一部分的函数有：

pair<node\*,node\*> build(const string &s,int bg,int ed);

int dfsPrint(node \*rt);

2.2 NFA到DFA

使用子集构造法构造DFA

函数：

void rmEps(node \*fa,node \*rt);

void NFA2DFA\_rmEps(node\* rt);

void getNodes(node \*rt);

void dfaPrint(int cur);

2.3 DFA最小化

void NFA2DFA(node \*rt);

void getMinDfa();

void printMinDfa(node \*bgNode);

void getCode(string filename,const string &regexString);