《编译原理课程实践》

实验报告

课　　程：	编译理论相关算法实现
学　　院：	计算机学院
专　　业：	计算机科学与技术
姓　　名：	梅瑞贤
学　　号：	21052021
老　　师：	黄孝喜
完成时间：	2023 年 12 月 28 日

目录

一、 词法分析器相关

1. 实验目的

掌握从正则表达式构造 NFA 的算法，掌握从 NFA 构造 DFA 的算法，掌握 DFA 最小化算法。

2. 实验内容

编写程序，从只含有星闭包、连接和或运算的正则表达式构造 NFA，并将 NFA 转换为 DFA，最后对 DFA 进行最小化处理。

3. 设计方案与算法描述

方案设计如下：

- 将输入的正则式转化为后缀表达式，并构后缀树
- 使用 Tompson 或者 Glushkov 算法将后缀树转化为 NFA
- 使用子集构造法将 NFA 转化为 DFA
- 使用 Hopcroft 算法对 DFA 进行最小化处理
- 使用 dot 语言输出 DFA 的图形化表示

Tompson 算法：

根据语法树构造 NFA，对于每个节点，如果是连接符，那么将左右子树的开始节点和结束节点分别连接起来，如果是或运算符，那么将左右子树的开始节点和结束节点分别连接到一个新的节点，如果是闭包运算符，那么将左子树的开始节点和结束节点分别连接到一个新的节点，然后将新的节点和自身连接起来，最后将新的节点和结束节点连接起来。

Glushkov 算法：

根据语法树构造 NFA，对于每个节点，维护开始集合、结束集合、关系集合和一个可空标志
构造 NFA 的过程中，对于每个节点，需要维护以下信息

- 开始集合：表示从该节点开始的所有可能的输入符号集合
- 结束集合：表示从该节点结束的所有可能的输入符号集合
- 关系集合：表示一个相邻的节点对
- 可空标志：表示该节点是否可以为空（即可以匹配空字符串）

对于不同的节点每种集合和标志的转移如下：

- 对于连接符，将左子树的开始集合和结束集合分别连接到右子树的开始集合和结束集合，如果左子树可空，那么将左子树的结束集合连接到右子树的开始集合

- 对于或运算符，将左子树的开始集合和结束集合分别连接到一个新的节点，将右子树的开始集合和结束集合分别连接到一个新的节点，然后将新的节点和自身连接起来，最后将新的节点和结束节点连接起来
- 对于闭包运算符，将左子树的开始集合和结束集合分别连接到一个新的节点，然后将新的节点和自身连接起来，最后将新的节点和结束节点连接起来
- 对于字符节点，将开始集合和结束集合分别连接到一个新的节点，然后将新的节点和结束节点连接起来
- 对于空节点，将开始集合和结束集合分别连接到一个新的节点，然后将新的节点和结束节点连接起来

最后根据根节点的关系集合构造 NFA，每个关系集合中的元素表示一个转换关系，其中第一个元素表示起始节点，第二个元素表示结束节点

根据根节点的开始集合和结束集合以及可空标志构造 NFA 的开始节点和结束节点

子集构造法：

使用 epsilon 闭包求出 NFA 的开始节点的 epsilon 闭包，作为 DFA 的开始节点

然后对于 DFA 的每个节点，对于每个输入符号，求出 NFA 中该节点的 epsilon 闭包经过该输入符号的转移，作为 DFA 中该节点经过该输入符号的转移

Hopcroft 算法：

Hopcroft 算法的核心思想是将 DFA 中的状态划分为等价类，然后将等价类作为新的状态，构造新的 DFA

Hopcroft 算法的过程如下：

- 将 DFA 中的状态划分为两个等价类，一个是终态，一个是非终态
- 对于每个等价类，对于每个输入符号，求出该等价类经过该输入符号的转移，如果该等价类经过该输入符号的转移后的状态不在该等价类中，那么将该等价类划分为两个等价类，一个是该等价类经过该输入符号的转移后的状态，一个是该等价类减去该等价类经过该输入符号的转移后的状态
- 重复上述过程，直到没有等价类发生变化
- 将等价类作为新的状态，构造新的 DFA
- 将新的 DFA 中的状态重新编号
- 将新的 DFA 中的终态设置为原来的终态
- 将新的 DFA 中的开始态设置为原来的开始态
- 将新的 DFA 中的状态按照编号排序

4. 测试结果

TestCase 1:

input:

```
(((a)b)c)|(as(b))
```

output:

```
digraph g {
	4 [shape=doublecircle];

	0 -> 1[label=a];
	1 -> 2[label=b];
	1 -> 3[label=s];
	2 -> 4[label=c];
	3 -> 4[label=b];
}
```

visual:

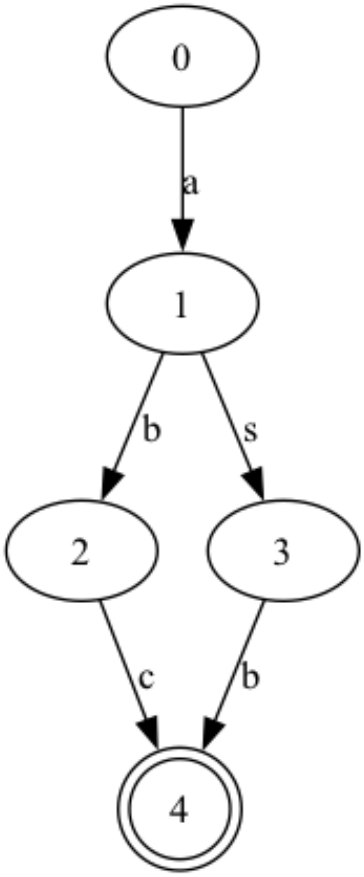

图 1: Test 1

TestCase 2:

input:

```
((((aa)aa)(aa)cc)a)|(aa*(ab*)*a)
```

output:

```
digraph g {
	2 [shape=doublecircle];
	3 [shape=doublecircle];
	5 [shape=doublecircle];
	6 [shape=doublecircle];
	7 [shape=doublecircle];
	8 [shape=doublecircle];
	11 [shape=doublecircle];

	0 -> 1[label=a];
	1 -> 2[label=a];
	2 -> 3[label=a];
	2 -> 4[label=b];
	3 -> 4[label=b];
	3 -> 5[label=a];
	4 -> 4[label=b];
	4 -> 6[label=a];
	5 -> 4[label=b];
	5 -> 7[label=a];
	6 -> 4[label=b];
	6 -> 6[label=a];
	7 -> 4[label=b];
	7 -> 8[label=a];
	8 -> 4[label=b];
	8 -> 6[label=a];
	8 -> 9[label=c];
	9 -> 10[label=c];
	10 -> 11[label=a];
}
```

visual:

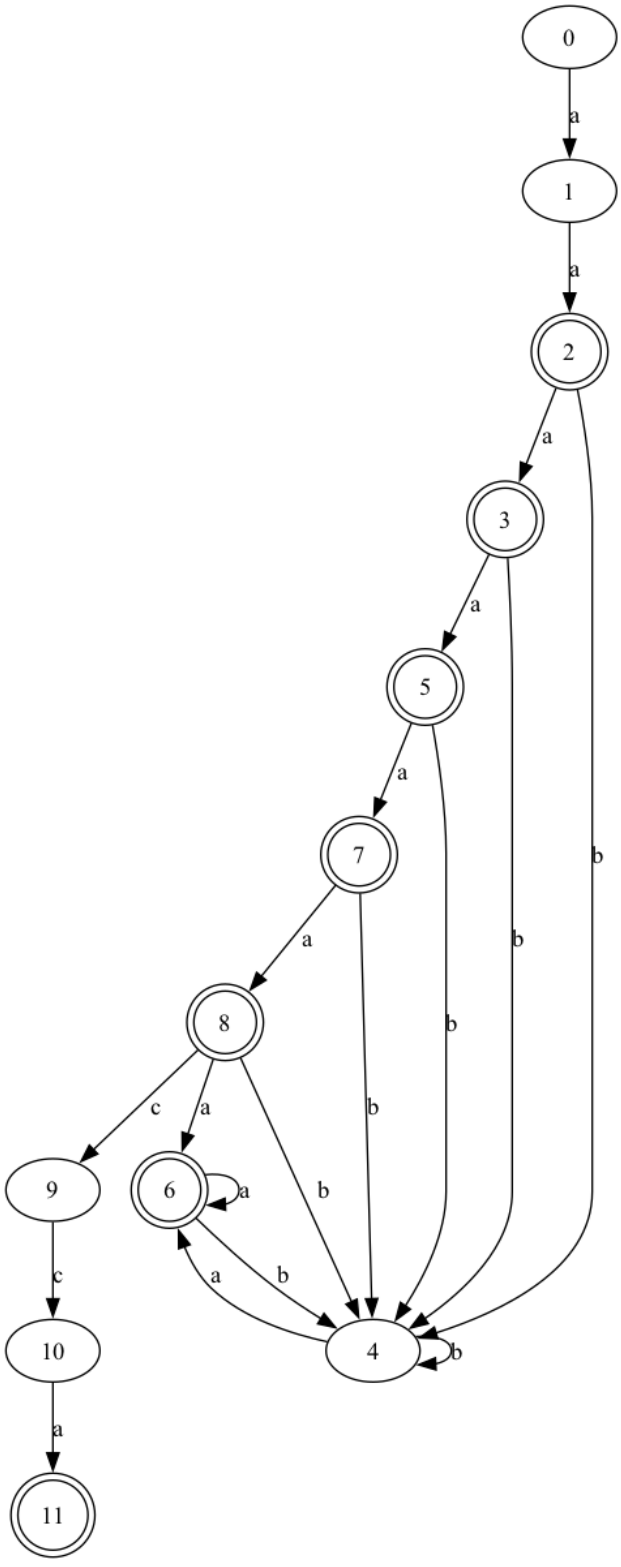

图 2: Test 2

TestCase 3:

input:

```
(aca(a((abb|ab)aa)(a*b)(aa))a)|(aa*(ab*)*a)
```

output:

```
digraph g {
	2 [shape=doublecircle];
	15 [shape=doublecircle];

	0 -> 1[label=a];
	1 -> 2[label=a];
	1 -> 3[label=c];
	2 -> 2[label=a];
	2 -> 4[label=b];
	3 -> 5[label=a];
	4 -> 2[label=a];
	4 -> 4[label=b];
	5 -> 6[label=a];
	6 -> 7[label=a];
	7 -> 8[label=b];
	8 -> 9[label=a];
	8 -> 10[label=b];
	9 -> 11[label=a];
	10 -> 9[label=a];
	11 -> 11[label=a];
	11 -> 12[label=b];
	12 -> 13[label=a];
	13 -> 14[label=a];
	14 -> 15[label=a];
}
```

visual:

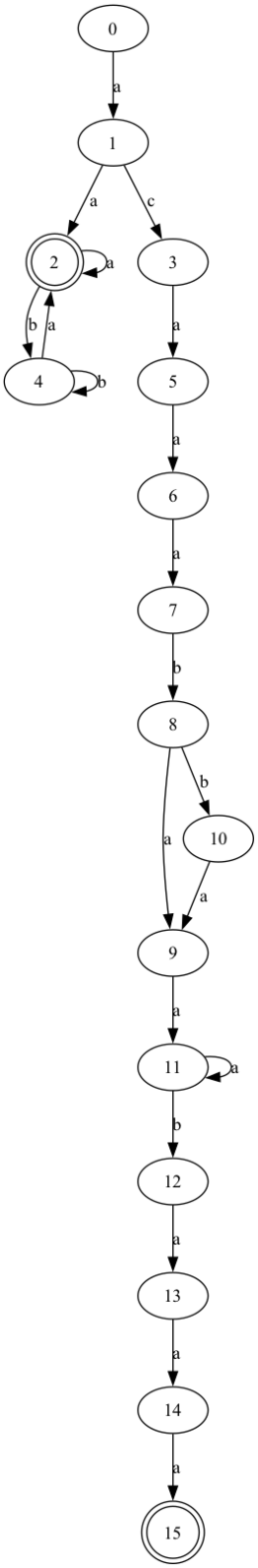

图 3: Test 3

TestCase 4:

input:

```
(a(xb)c)a*ac(ba)c|ac(cad)*|a|av
```

output:

```
digraph g {
	1 [shape=doublecircle];
	2 [shape=doublecircle];
	3 [shape=doublecircle];

	0 -> 1[label=a];
	1 -> 2[label=c];
	1 -> 3[label=v];
	1 -> 4[label=x];
	2 -> 5[label=c];
	4 -> 6[label=b];
	5 -> 7[label=a];
	6 -> 8[label=c];
	7 -> 2[label=d];
	8 -> 9[label=a];
	9 -> 9[label=a];
	9 -> 10[label=c];
	10 -> 11[label=b];
	11 -> 12[label=a];
	12 -> 3[label=c];
}
```

visual:

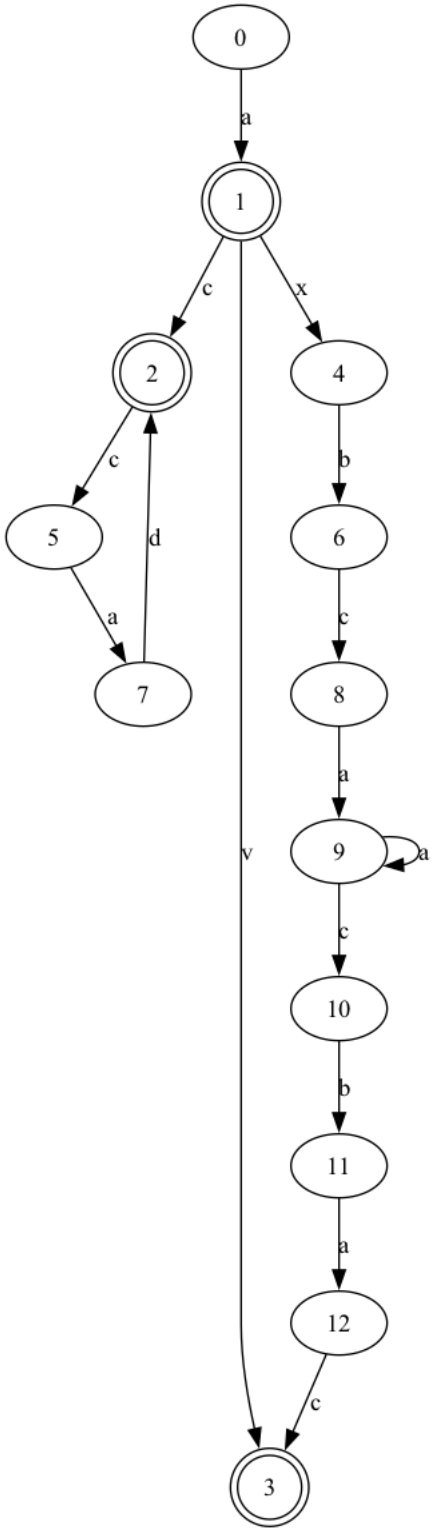

图 4: Test 4

TestCase 5:

input:

```
(a(ab)*)*|(ba)*
```

output:

```
digraph g {
	0 [shape=doublecircle];
	1 [shape=doublecircle];
	3 [shape=doublecircle];
	4 [shape=doublecircle];

	0 -> 1[label=a];
	0 -> 2[label=b];
	1 -> 3[label=a];
	2 -> 4[label=a];
	3 -> 1[label=b];
	3 -> 3[label=a];
	4 -> 2[label=b];
}
```

visual:

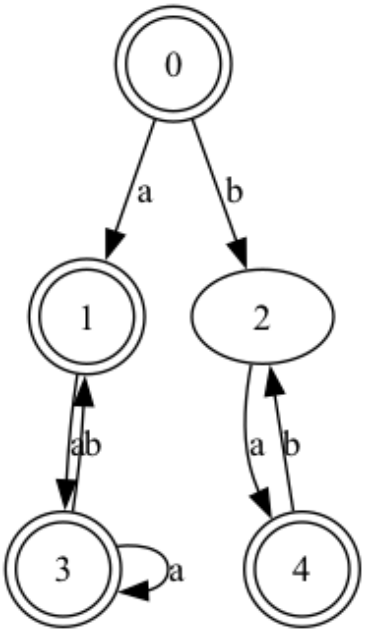

图 5: Test 5

5. 源代码

```cpp
#include <bits/stdc++.h>

using i64 = long long;

bool is_op(char x) {
    return x == '.' || x == '*' || x == '|';
}

int priority(char x) {
    if (x == '(') return 0;
    else if (x == '|') return 1;
    else if (x == '.') return 2;
    else if (x == '*') return 3;
    else return 100;
}

// 将表达式处理成后缀形式方便建立语法树
std::vector<char> suf_exp(std::string s) {
    // 添加隐含的连接符
    std::vector<char> ch;

    bool begin = true;
    for (auto x : s) {
        if (x == '(') {
            if (!begin) {
                ch.push_back('.');
            }
            begin = true;
        } else if (x == '|') {
            begin = true;
        } else if (x == ')' || x == '*') {
            begin = false;
        } else {
            if (!begin) {
                ch.push_back('.');
            }
            begin = false;
        }
        ch.push_back(x);
    }

    //通过栈来处理括号，优先级
    std::stack<char> op;
    std::vector<char> exp;

    for (auto x : ch) {
        if (is_op(x)) {
            if (op.empty() || priority(op.top()) < priority(x)) {
```

```
49              op.push(x);
50          } else {
51              while (!op.empty() && priority(op.top()) >= priority(x)) {
52                  exp.push_back(op.top());
53                  op.pop();
54              }
55              op.push(x);
56          }
57      } else if (x == '(') {
58          op.push(x);
59      } else if (x == ')') {
60          bool ok = false;
61          while (!op.empty()) {
62              auto t = op.top();
63              op.pop();
64              if (t == '(') {
65                  ok = true;
66                  break;
67              } else {
68                  exp.push_back(t);
69              }
70          }
71          if (!ok) {
72              std::cerr << "Invalid input!\n";
73              exit(1);
74          }
75      } else {
76          exp.push_back(x);
77      }
78  }
79
80  while (!op.empty()) {
81      if (op.top() == '(') {
82          std::cerr << "Invalid input!\n";
83          exit(1);
84      }
85      exp.push_back(op.top());
86      op.pop();
87  }
88
89  return exp;
90 }
91
92 struct GrammTree {
93  std::vector<char> node;
94  std::vector<std::vector<int>> adj;
95
96  int n;
97
98  GrammTree() :n(0) {}
```

```
99
100     int new_node(char x) {
101         node.push_back(x);
102         adj.emplace_back();
103         return n++;
104     };
105
106     void show() {
107         std::cout << "digraph g {\n";
108         for (int i = 0; i < n; ++i) {
109             std::cout << "\t" << i << " [label=\"" << node[i] << "\"];\n";
110         }
111         std::cout << "\n";
112         for (int i = 0; i < n; ++i) {
113
114             for (auto x : adj[i]) {
115                 std::cout << "\t" << i << " -> " << x << ";\n";
116             }
117         }
118         std::cout << "}\n";
119     }
120 };
121
122 // 根据运算规则和后缀表达式建立语法树
123 GrammTree build_tree(std::vector<char> suf) {
124     GrammTree T;
125
126     std::vector<int> roots;
127
128     for (auto x : suf) {
129         if (x == '*') {
130             int now = T.new_node(x);
131
132             assert(!roots.empty());
133
134             auto son = roots.back();
135
136             T.adj[now].push_back(son);
137             roots.pop_back();
138             roots.push_back(now);
139         } else if (x == '.' || x == '|') {
140             int now = T.new_node(x);
141
142             assert(roots.size() >= 2);
143
144             for (int i = 0; i < 2; ++i) {
145                 T.adj[now].push_back(roots[roots.size() - 2 + i]);
146             }
147             roots.pop_back();
148             roots.pop_back();
```

```
149             roots.push_back(now);
150         } else {
151             int now = T.new_node(x);
152             roots.push_back(now);
153         }
154     }
155 
156     return T;
157 }
158 
159 
160 struct automaton {
161     std::vector<std::vector<std::pair<int, int>>> adj;
162 
163     std::set<int> begin, end, alphabet;
164 
165     int n;
166 
167     automaton() :n(0), begin(), end() {}
168     automaton(int n) :n(n), begin(), end(), adj(n) {}
169 
170     // 新建一个节点
171     int NEW() {
172         adj.emplace_back();
173         return n++;
174     }
175 
176     // 添加一条边
177     void add_edge(int u, int v, int c) {
178         if (c >= 0) alphabet.insert(c);
179         adj[u].emplace_back(v, c);
180     }
181 
182     // 删除重边
183     void unique() {
184         for (int i = 0; i < n; ++i) {
185             std::sort(adj[i].begin(), adj[i].end());
186             adj[i].erase(std::unique(adj[i].begin(), adj[i].end()), adj[i].end());
187         }
188     }
189 
190     // 使用dot语言输出
191     void show() {
192         unique();
193         std::cout << "digraph g {\n";
194         for (auto x : end) {
195             std::cout << "\t" << x << " [shape=doublecircle];\n";
196         }
197         std::cout << "\n";
198         for (int i = 0; i < n; ++i) {
```

```
199             for (auto [x, lable] : adj[i]) {
200                 std::cout << "\t" << i << " -> " << x << "[label=" << (lable >= 0 ? std::string(1,
                        char(lable)) : std::string("eps"))
201                 << "]" << ";\n";
202             }
203         }
204         std::cout << "}\n";
205     }
206 
207     bool match (std::string s) {
208         std::set<int> cur = begin;
209         for (auto x : s) {
210             std::set<int> nxt;
211             for (auto u : cur) {
212                 for (auto [v, c] : adj[u]) {
213                     if (c == x) {
214                         nxt.insert(v);
215                     }
216                 }
217             }
218             cur = nxt;
219         }
220         for (auto x : cur) {
221             if (end.count(x)) {
222                 return true;
223             }
224         }
225         return false;
226     }
227 };
228 
229 // Thompson算法
230 automaton Tompson(std::vector<char> suf) {
231     automaton nfa;
232     auto T = build_tree(suf);
233 
234     int root = T.n - 1;
235 
236     nfa.NEW();
237 
238     auto dfs = [&] (auto dfs, int x, int begin) -> int {
239         if (T.node[x] == '|') {
240             int end = nfa.NEW();
241             for (auto son : T.adj[x]) {
242                 int son_begin = nfa.NEW();
243                 int son_end = dfs(dfs, son, son_begin);
244                 nfa.add_edge(begin, son_begin, -1);
245                 nfa.add_edge(son_end, end, -1);
246             }
247             return end;
```

```
248         } else if (T.node[x] == '*') {
249             int son = T.adj[x][0];
250             int son_begin = nfa.NEW();
251 
252             int son_end = dfs(dfs, son, son_begin);
253 
254             int end = nfa.NEW();
255 
256             nfa.add_edge(begin, son_begin, -1);
257             nfa.add_edge(begin, end, -1);
258             nfa.add_edge(son_end, son_begin, -1);
259             nfa.add_edge(son_end, end, -1);
260 
261             return end;
262         } else if (T.node[x] == '.') {
263             int ls = T.adj[x][0], rs = T.adj[x][1];
264             int rbegin = dfs(dfs, ls, begin);
265             return dfs(dfs, rs, rbegin);
266         } else {
267             int end = nfa.NEW();
268             nfa.add_edge(begin, end, T.node[x]);
269             return end;
270         }
271     };
272 
273     int end = dfs(dfs, root, 0);
274     nfa.begin.insert(0);
275     nfa.end.insert(end);
276     return nfa;
277 }
278 
279 // Glushkov算法
280 automaton Glushkov(std::vector<char> suf) {
281     automaton nfa;
282     auto T = build_tree(suf);
283     int root = T.n - 1;
284 
285 
286     std::vector<int> new_node(T.n);
287 
288     std::map<int, char> mp;
289 
290     {
291         int x = 0;
292         for (int i = 0; i < T.n; ++i) {
293             if (is_op(T.node[i])) {
294                 new_node[i] = T.node[i];
295             } else {
296                 mp[x] = T.node[i];
297                 new_node[i] = x++;
```

```
298             }
299         }
300         nfa = automaton(x);
301     }
302
303
304     auto dfs = [&] (auto dfs, int x) -> std::tuple<std::set<int>, std::set<int>, std::set<std::
            array<int,2>>, bool> {
305         if (new_node[x] == '|') {
306             auto [Pe, De, Fe, Ae] = dfs(dfs, T.adj[x][0]);
307             auto [Pf, Df, Ff, Af] = dfs(dfs, T.adj[x][1]);
308             bool A = Ae || Af;
309             auto P = Pe;
310             for (auto x : Pf) {
311                 P.insert(x);
312             }
313
314             auto D = De;
315             for (auto x : Df) {
316                 D.insert(x);
317             }
318
319             auto F = Fe;
320             for (auto x : Ff) {
321                 F.insert(x);
322             }
323
324             return {P, D, F, A};
325         } else if (new_node[x] == '*') {
326             auto [P, D, F, A] = dfs(dfs, T.adj[x][0]);
327             auto NF = F;
328             for (auto a : D) {
329                 for (auto b : P) {
330                     NF.insert({a, b});
331                 }
332             }
333             return {P, D, NF, true};
334         } else if (new_node[x] == '.') {
335             auto [Pe, De, Fe, Ae] = dfs(dfs, T.adj[x][0]);
336             auto [Pf, Df, Ff, Af] = dfs(dfs, T.adj[x][1]);
337
338             auto P = Pe;
339             if (Ae) {
340                 for (auto x : Pf) {
341                     P.insert(x);
342                 }
343             }
344
345             auto D = Df;
346             if (Af) {
```

```
347                 for (auto x : De) {
348                     D.insert(x);
349                 }
350             }
351
352             auto F = Fe;
353             for (auto x : Ff) {
354                 F.insert(x);
355             }
356             for (auto x : De) {
357                 for (auto y : Pf) {
358                     F.insert({x, y});
359                 }
360             }
361
362             return {P, D, F, Ae && Af};
363         } else {
364             return {std::set<int>({new_node[x]}), std::set<int>({new_node[x]}), std::set<std::array<
                    int,2>>({}), false};
365         }
366     };
367
368     auto [P, D, F, A] = dfs(dfs, root);
369
370     int start = nfa.NEW();
371     nfa.begin.insert(start);
372     nfa.end = D;
373     if (A) nfa.end.insert(start);
374
375     for (auto x : P) {
376         nfa.add_edge(start, x, mp[x]);
377     }
378
379     for (auto [x, y] : F) {
380         nfa.add_edge(x, y, mp[y]);
381     }
382
383     return nfa;
384 }
385
386 // 子集构造法
387 automaton nfatodfa(automaton nfa) {
388     automaton dfa;
389
390     std::map<std::set<int>, int> mp;
391     std::vector<std::set<int>> sets;
392
393     // epsilon闭包
394     auto epsilon = [&] (std::set<int> t) {
395         std::queue<int> q;
```

```
396         std::set<int> ans;
397 
398         for (auto x : t) {
399             q.push(x);
400             ans.insert(x);
401         }
402 
403         while (!q.empty()) {
404             auto x = q.front();
405             q.pop();
406 
407             for (auto [x, v] : nfa.adj[x]) {
408                 // epislon
409                 if (v == -1) {
410                     if (!ans.count(x)) {
411                         ans.insert(x);
412                         q.push(x);
413                     }
414                 }
415             }
416         }
417 
418         return ans;
419     };
420 
421     // 集合t经过字符c的转移
422     auto move = [&] (std::set<int> t, int c) {
423         std::set<int> ans;
424 
425         for (auto x : t) {
426             for (auto [y, v] : nfa.adj[x]) {
427                 if (v == c) {
428                     ans.insert(y);
429                 }
430             }
431         }
432 
433         return ans;
434     };
435 
436     std::set<int> cur = epsilon(nfa.begin);
437 
438     // 将获得的集合作为dfa的新状态
439     auto ins = [&] (const std::set<int>& t) {
440         if (!mp.count(t)) {
441             sets.push_back(t);
442             mp[t] = dfa.NEW();
443             for (auto x : t) {
444                 if (nfa.end.count(x)) {
445                     dfa.end.insert(mp[t]);
```

```
446                 }
447                 if (nfa.begin.count(x)) {
448                     dfa.begin.insert(mp[t]);
449                 }
450             }
451         }
452     };
453 
454     ins(cur);
455 
456     int i = 0;
457     while (i < dfa.n) {
458         cur = sets[i];
459         for (auto x : nfa.alphabet) {
460             auto t = epsilon(move(cur, x));
461             if (t.empty()) continue;
462             ins(t);
463             dfa.add_edge(i, mp[t], x);
464         }
465         i++;
466     }
467     return dfa;
468 }
469 
470 // 集合交集
471 std::set<int> operator & (const std::set<int> a, const std::set<int> b) {
472     std::set<int> ans;
473     for (auto x : a) {
474         if (b.count(x)) {
475             ans.insert(x);
476         }
477     }
478     return ans;
479 }
480 
481 // 集合并集
482 std::set<int> operator + (const std::set<int> a, const std::set<int> b) {
483     std::set<int> ans;
484     for (auto x : a) {
485         ans.insert(x);
486     }
487     for (auto x : b) {
488         ans.insert(x);
489     }
490     return ans;
491 }
492 
493 // 集合减法
494 std::set<int> operator - (const std::set<int> a, const std::set<int> b) {
495     std::set<int> ans(a);
```

```
496     for (auto x : b) {
497         ans.erase(x);
498     }
499     return ans;
500 }
501 
502 // Hopcroft算法
503 automaton Hopcroft(automaton dfa) {
504     std::set<std::set<int>> P, W;
505     int n;
506     n = dfa.n;
507     std::set<int> F(dfa.end.begin(), dfa.end.end());
508 
509     std::vector<std::vector<std::pair<int,int>>> adj_rev(n);
510 
511     for (int i = 0; i < n; ++i) {
512         for (auto [x, v] : dfa.adj[i]) {
513             adj_rev[x].emplace_back(i, v);
514         }
515     }
516 
517     P.insert(F);
518     std::set<int> rem_f;
519     for (int i = 0; i < n; ++i) {
520         if (!F.count(i)) {
521             rem_f.insert(i);
522         }
523     }
524     P.insert(rem_f);
525 
526     W.insert(F);
527 
528     while (!W.empty()) {
529         auto A = *W.begin();
530         W.erase(W.begin());
531         for (auto c : dfa.alphabet) {
532             auto prev = [&] (const std::set<int>& t, int c) {
533                 std::set<int> ans;
534                 for (auto x : t) {
535                     for (auto [y, v] : adj_rev[x]) {
536                         if (v == c) {
537                             ans.insert(y);
538                         }
539                     }
540                 }
541                 return ans;
542             };
543 
544             auto X = prev(A, c);
545 
```

```
546             if (X.empty()) continue;
547
548             auto tmpp = P;
549             for (auto Y : tmpp) {
550                 auto Y1 = Y & X;
551                 auto Y2 = Y - X;
552                 if (Y1.empty() || Y2.empty()) continue;
553                 P.erase(Y);
554                 P.insert(Y1);
555                 P.insert(Y2);
556                 if (W.count(Y)) {
557                     W.erase(Y);
558                     W.insert(Y1);
559                     W.insert(Y2);
560                 } else {
561                     if (Y1.size() <= Y2.size()) {
562                         W.insert(Y1);
563                     } else {
564                         W.insert(Y2);
565                     }
566                 }
567             }
568         }
569     }
570
571     //rebuild
572     automaton dfa_min;
573     std::map<std::set<int>, int> mp;
574     auto get_id = [&] (const std::set<int>& t) {
575         if (!mp.count(t)) {
576             mp[t] = dfa_min.NEW();
577             for (auto x : t) {
578                 if (dfa.end.count(x)) {
579                     dfa_min.end.insert(mp[t]);
580                 }
581                 if (dfa.begin.count(x)) {
582                     dfa_min.begin.insert(mp[t]);
583                 }
584             }
585         }
586         return mp[t];
587     };
588
589     std::vector<int> belong(n);
590     for (auto x : P) {
591         int id = get_id(x);
592         for (auto y : x) {
593             belong[y] = id;
594         }
595     }
```

```
596
597     for (int x = 0; x < n; ++x) {
598         for (auto [y, c] : dfa.adj[x]) {
599             auto u = belong[x], v = belong[y];
600             dfa_min.add_edge(u, v, c);
601         }
602     }
603
604     return dfa_min;
605 }
606
607 automaton exptoam(std::string s) {
608
609     auto nfa = Glushkov(suf_exp(s));
610     // nfa.show();
611
612     auto dfa = nfatodfa(nfa);
613     // dfa.show();
614
615     auto min = Hopcroft(dfa);
616
617     return min;
618 }
619
620 int main() {
621     std::string s;
622     std::cin >> s;
623     auto dfa_min = exptoam(s);
624     dfa_min.show();
625
626     return 0;
627 }
```

二、 LL1 文法相关

1. 实验目的

掌握从 LL1 文法提取公共因子，消除左递归的算法，掌握 LL1 文法的 FIRST 集和 FOLLOW 集的求法，掌握 LL1 文法的预测分析表的构造算法。

2. 实验内容

编写程序，输入上下文无关文法，如果可以化简为 LL1 文法则输出提取公共左因子后的文法，消除左递归后的文法，FIRST 集，FOLLOW 集，预测分析表，否则报错。

3. 设计方案与算法描述

方案设计如下：

- 使用暴力进行字符串切割和匹配，将输入的文法转化为 token 序列
- 使用 trie 树和 dfs 提取文法的左公共因子
- 对提取完左公共因子的文法消去左递归
- 根据定义求出 FIRST 集、FOLLOW 集、预测分析表
- 根据预测分析表进行语法分析

根据 trie 树提取左公共因子：

经过观察，把文法全部插入到 trie 树中，然后对于每个节点，如果该节点的子节点数大于 1，那么就将该节点的子节点的公共前缀作为公共因子，然后将该节点的子节点的公共前缀作为新的节点，将该节点的子节点的公共前缀作为新的子节点插入到新的节点中，然后将该节点的子节点的公共前缀从该节点的子节点中删除，最后将该节点的子节点的公共前缀作为新的节点的子节点。

消除左递归：

上课详细讲过，不多赘述

求 FIRST 等集合：

根据定义求出 FIRST 集、FOLLOW 集、预测分析表

4. 测试结果

TestCase 1:

input:

```
E -> E+T | E-T | T
T -> T*F | T/F | F
F -> (E) | i
```

output:

```
E -> E + T | E - T | T
T -> T * F | T / F | F
F -> ( E ) | i

After remove left recursion:
E -> T tmp0
T -> F tmp1
F -> ( E ) | i
tmp0 -> ε| + T tmp0 | - T tmp0
tmp1 -> ε| * F tmp1 | / F tmp1

First:ε
: ε
E: ( i
T: ( i
F: ( i
+: +
-: -
*: *
/: /
(: (
): )
i: i
tmp0: ε+ -
tmp1: ε* /

FOLLOW:ε
:
E: $ )
T: $ + - )
F: $ + - * / )
+:
-:
*:
/:
(:
):
i:
tmp0: $ )
tmp1: $ + - )
```

```
41
42 SELECT:
43 E -> T tmp0 : ( i
44 T -> F tmp1 : ( i
45 F -> ( E ) : (
46 F -> i : i
47 tmp0 -> ε: $ )
48 tmp0 -> + T tmp0 : +
49 tmp0 -> - T tmp0 : -
50 tmp1 -> ε: $ + - )
51 tmp1 -> * F tmp1 : *
52 tmp1 -> / F tmp1 : /
53 LL1
```

graph:

	$	ε	+	-	*	/	(	)	i
E							E -> T tmp0		E -> T tmp0
T							T -> F tmp1		T -> F tmp1
F							F -> (E)		F -> i
tmp0	tmp0 -> ε		tmp0 -> + T tmp0	tmp0 -> - T tmp0				tmp0 -> ε	
tmp1	tmp1 -> ε		tmp1 -> ε	tmp1 -> ε	tmp1 -> * F tmp1	tmp1 -> / F tmp1		tmp1 -> ε	

TestCase 2:

input:

```
S -> Pe | c
P -> DE | f
D -> SP | g
E -> abs
```

output:

```
S -> P e | c
P -> D E | f
D -> S P | g
E -> a b s

After remove left recursion:
S -> P e | c
P -> D E | f
D -> c P tmp0 | f e P tmp0 | g tmp0
E -> a b s
tmp0 -> ε| E e P tmp0

First:ε
: ε
S: c f g
P: c f g
D: c f g
E: a
e: e
c: c
f: f
g: g
a: a
b: b
s: s
tmp0: εa

FOLLOW:ε
:
S: $
P: e a
D: a
E: e a
e:
c:
f:
g:
a:
b:
s:
tmp0: a
```

```
42
43 SELECT:
44 S -> P e : c f g
45 S -> c : c
46 P -> D E : c f g
47 P -> f : f
48 D -> c P tmp0 : c
49 D -> f e P tmp0 : f
50 D -> g tmp0 : g
51 E -> a b s : a
52 tmp0 -> ε: a
53 tmp0 -> E e P tmp0 : a
54 Not LL1
```

TestCase 3:

input:

```
A -> abcde | abcdf | abcdg | abcdef | abce | abcd | abbe | abbc | abcc | aabd
```

output:

```
A -> a a b d | a b b c | a b b e | a b c c | a b c d | a b c d e | a b c d e f | a b c d f | a b c
    d g | a b c e

After remove left recursion:
A -> a tmp5
tmp0 -> c | e
tmp1 -> ε| f
tmp2 -> ε| e tmp1 | f | g
tmp3 -> c | d tmp2 | e
tmp4 -> b tmp0 | c tmp3
tmp5 -> a b d | b tmp4

First:ε
: ε
A: a
a: a
b: b
c: c
d: d
e: e
f: f
g: g
tmp0: c e
tmp1: εf
tmp2: εe f g
tmp3: c d e
tmp4: b c
tmp5: a b

FOLLOW:ε
:
A: $
a:
b:
c:
d:
e:
f:
g:
tmp0: $
tmp1: $
tmp2: $
tmp3: $
tmp4: $
```

```
tmp5: $

SELECT:
A -> a tmp5 : a
tmp0 -> c : c
tmp0 -> e : e
tmp1 -> ε: $
tmp1 -> f : f
tmp2 -> ε: $
tmp2 -> e tmp1 : e
tmp2 -> f : f
tmp2 -> g : g
tmp3 -> c : c
tmp3 -> d tmp2 : d
tmp3 -> e : e
tmp4 -> b tmp0 : b
tmp4 -> c tmp3 : c
tmp5 -> a b d : a
tmp5 -> b tmp4 : b
LL1
```

graph:

	$	ε	a	b	c	d	e	f	g
A			A -> a tmp5						
tmp0					tmp0 -> c		tmp0 -> e		
tmp1	tmp1 -> ε							tmp1 -> f	
tmp2	tmp2 -> ε						tmp2 -> e tmp1	tmp2 -> f	tmp2 -> g
tmp3					tmp3 -> c	tmp3 -> d tmp2	tmp3 -> e		
tmp4				tmp4 -> b tmp0	tmp4 -> c tmp3				
tmp5			tmp5 -> a b d	tmp5 -> b tmp4					

TestCase 4:

input:

```
A -> Bb | c
B -> ε| d
```

output:

```
A -> B b | c
B -> ε| d

After remove left recursion:
A -> B b | c
B -> ε| d

First:ε
: ε
A: b c d
B: ε d
b: b
c: c
d: d

FOLLOW:ε
:
A: $
B: b
b:
c:
d:

SELECT:
A -> B b : b d
A -> c : c
B -> ε: b
B -> d : d
LL1
```

graph:

	$	ε	b	c	d
A			A -> B b	A -> c	A -> B b
B			B -> ε		B -> d

TestCase 5:

input:

```
A -> BC | a
B -> ε| b
C -> c | ε
```

output:

```
A -> B C | a
B -> ε| b
C -> ε| c

After remove left recursion:
A -> B C | a
B -> ε| b
C -> ε| c

First:ε
: ε
A: ε a b c
B: ε b
C: ε c
a: a
b: b
c: c

FOLLOW:ε
:
A: $
B: $ c
C: $
a:
b:
c:

SELECT:
A -> B C : $ b c
A -> a : a
B -> ε: $ c
B -> b : b
C -> ε: $
C -> c : c
LL1
```

graph:

	$	ε	a	b	c
A	A -> B C		A -> a	A -> B C	A -> B C
B	B -> ε			B -> b	B -> ε
C	C -> ε				C -> c

TestCase 6:

input:

```
A -> B
B -> aB | b
```

output:

```
A -> B
B -> a B | b

After remove left recursion:
A -> B
B -> a B | b

First:ε
: ε
A: a b
B: a b
a: a
b: b

FOLLOW:ε
:
A: $
B: $
a:
b:

SELECT:
A -> B : a b
B -> a B : a
B -> b : b
LL1
```

graph:

	$	ε	a	b
A			A -> B	A -> B
B			B -> a B	B -> b

TestCase 7:

input:

```
A -> a | B
B -> b | ε
```

output:

```
A -> B | a
B -> ε| b

After remove left recursion:
A -> B | a
B -> ε| b

First:ε
: ε
A: ε a b
B: ε b
a: a
b: b

FOLLOW:ε
:
A: $
B: $
a:
b:

SELECT:
A -> B : $ b
A -> a : a
B -> ε: $
B -> b : b
LL1
```

graph:

	$	ε	a	b
A	A -> B		A -> a	A -> B
B	B -> ε			B -> b

TestCase 8:

input:

```
A -> BC
B -> b | ε
C -> c
```

output:

```
A -> B C
B -> ε| b
C -> c

After remove left recursion:
A -> B C
B -> ε| b
C -> c

First:ε
: ε
A: b c
B: ε b
C: c
b: b
c: c

FOLLOW:ε
:
A: $
B: c
C: $
b:
c:

SELECT:
A -> B C : b c
B -> ε: c
B -> b : b
C -> c : c
LL1
```

graph:

	$	ε	b	c
A			A -> B C	A -> B C
B			B -> b	B -> ε
C				C -> c

TestCase 9:

input:

```
A → aB
B → b | ε
```

output:

```
A -> a B
B -> ε| b

After remove left recursion:
A -> a B
B -> ε| b

First:ε
: ε
A: a
B: ε b
a: a
b: b

FOLLOW:ε
:
A: $
B: $
a:
b:

SELECT:
A -> a B : a
B -> ε: $
B -> b : b
LL1
```

graph:

	$	ε	a	b
A			A -> a B	
B	B -> ε			B -> b

TestCase 10:

input:

```
A → aABe | a
B → Bb | d
```

output:

```
A -> a | a A B e
B -> B b | d

After remove left recursion:
A -> a tmp1
B -> d tmp0
tmp0 -> ε| b tmp0
tmp1 -> ε| A B e

First:ε
: ε
A: a
B: d
a: a
e: e
b: b
d: d
tmp0: εb
tmp1: εa

FOLLOW:ε
:
A: $ d
B: e
a:
e:
b:
d:
tmp0: e
tmp1: $ d

SELECT:
A -> a tmp1 : a
B -> d tmp0 : d
tmp0 -> ε: e
tmp0 -> b tmp0 : b
tmp1 -> ε: $ d
tmp1 -> A B e : a
LL1
```

graph:

	$	ε	a	e	b	d
A			A -> a tmp1			
B						B -> d tmp0
tmp0				tmp0 -> ε	tmp0 -> b tmp0	
tmp1	tmp1 -> ε		tmp1 -> A B e			tmp1 -> ε

TestCase 11:

input:

```
int ->
float ->
id ->
D → T V
T → int | float
V → id,V | id
```

output:

```
D -> T
T -> int | float
V -> id | id , V

After remove left recursion:
D -> T
T -> int | float
V -> id tmp0
tmp0 -> ε| , V

First:ε
: ε
int: int
float: float
id: id
D: int float
T: int float
V: id
,: ,
tmp0: ε,

FOLLOW:ε
:
int: $
float:
id:
D:
T:
V:
,:
tmp0:

SELECT:
D -> T : int float
T -> int : int
T -> float : float
V -> id tmp0 : id
tmp0 -> ε:
tmp0 -> , V : ,
```

```
40 LL1
```

graph:

	$	ε	int	float	id	,
D			D -> T	D -> T		
T			T -> int	T -> float		
V					V -> id tmp0	
tmp0						tmp0 -> , V

TestCase 12:

input:

```
S → a | ^ | (T)
T → T,S | S
```

output:

```
S -> a | ^ | ( T )
T -> S | T , S

After remove left recursion:
S -> a | ^ | ( T )
T -> a tmp0 | ^ tmp0 | ( T ) tmp0
tmp0 -> ε| , S tmp0

First:ε
: ε
S: a ^ (
T: a ^ (
a: a
^: ^
(: (
): )
,: ,
tmp0: ε,

FOLLOW:ε
:
S: $ ) ,
T: )
a:
^:
(:
):
,:
tmp0: )

SELECT:
S -> a : a
S -> ^ : ^
S -> ( T ) : (
T -> a tmp0 : a
T -> ^ tmp0 : ^
T -> ( T ) tmp0 : (
tmp0 -> ε: )
tmp0 -> , S tmp0 : ,
LL1
```

graph:

	$	ε	a		^ (	)	,
S			S -> a	S ->	S -> (T)		
T			T -> a tmp0	T -> tmp0	T -> (T) tmp0		
tmp0						tmp0 -> ε	tmp0 -> , S tmp0

5. 源代码

```cpp
#include <bits/stdc++.h>

using i64 = long long;

// 集合交集
std::set<int> operator & (const std::set<int> a, const std::set<int> b) {
    std::set<int> ans;
    for (auto x : a) {
        if (b.count(x)) {
            ans.insert(x);
        }
    }
    return ans;
}

// 集合并集
std::set<int> operator + (const std::set<int> a, const std::set<int> b) {
    std::set<int> ans;
    for (auto x : a) {
        ans.insert(x);
    }
    for (auto x : b) {
        ans.insert(x);
    }
    return ans;
}

// 集合减法
std::set<int> operator - (const std::set<int> a, const std::set<int> b) {
    std::set<int> ans(a);
    for (auto x : b) {
        ans.erase(x);
    }
    return ans;
}

struct rule {
    std::string L;
    std::vector<std::string> R;

    rule(const std::string s) {
        std::stringstream ss(s);
        ss >> L;
        std::string tmp, sep;
        while (ss >> sep >> tmp) {
            R.push_back(tmp);
        }
    }
```

```
49
50      void show() {
51          std::cout << L << " =: ";
52          for (auto &s : R) {
53              std::cout << s << " ";
54          }
55          std::cout << std::endl;
56      }
57  };
58
59  struct rule_token {
60      int L;
61      std::vector<std::vector<int>> R;
62  };
63
64  struct trie{
65      std::map<int, trie*> nxt;
66      bool is_end = false;
67      int count = 0;
68      int id = -1;
69
70      void insert(std::vector<int> &s, int pos) {
71          if (pos == s.size()) {
72              is_end = true;
73              return;
74          }
75          if (!nxt.count(s[pos])) {
76              nxt[s[pos]] = new trie();
77          }
78          nxt[s[pos]]->count++;
79          nxt[s[pos]]->id = s[pos];
80          nxt[s[pos]]->insert(s, pos + 1);
81      }
82
83      ~trie() {
84          for (auto &[c, p] : nxt) {
85              delete p;
86          }
87      }
88  };
89
90  struct gramma_token{
91      std::map<int, std::vector<std::vector<int>>> G;
92      std::vector<std::string> mp;
93      int n = 0, tmp_cnt = 0;
94      int start = 1;
95
96      std::map<int, std::set<int>> first, follow;
97
98      std::map<int, std::vector<std::set<int>>> select;
```

```
99
100     void add_rule(int l, std::vector<int> r) {
101         G[l].push_back(r);
102     }
103
104     int new_rule() {
105         return n++;
106     }
107
108
109     void add_tmp () {
110         while (mp.size() < n) {
111             mp.push_back("tmp" + std::to_string(tmp_cnt++));
112         }
113     }
114
115     void unique() {
116         for (auto &[l, r] : G) {
117             std::sort(r.begin(), r.end());
118             r.erase(std::unique(r.begin(), r.end()), r.end());
119         }
120     }
121
122     void remove_rep() {
123         bool modify = true;
124         while (modify) {
125             modify = false;
126             auto G_T = G;
127             for (auto &[l, r] : G_T) {
128                 trie t;
129
130                 for (auto x : r) {
131                     t.insert(x, 0);
132                 }
133
134                 auto dfs = [&] (auto dfs, trie* cur) -> std::vector<int> {
135                     if (cur->count == 1 && cur->is_end) {
136                         return std::vector<int> ();
137                     }
138
139                     std::vector<std::vector<int>> sons;
140
141                     if (cur->is_end) {
142                         sons.push_back(std::vector<int> (1, 0));
143                     }
144
145                     for (auto &[c, p] : cur->nxt) {
146                         auto s = dfs(dfs, p);
147                         s.push_back(c);
148                         sons.push_back(s);
```

```
149                 }
150
151                 if (cur == &t) {
152                     G[l].clear();
153                     for (auto x : sons) {
154                         std::reverse(x.begin(), x.end());
155                         G[l].push_back(x);
156                     }
157                     return std::vector<int> ();
158                 }
159                 if (sons.size() > 1) {
160                     modify = true;
161                     int x = new_rule();
162                     for (auto T : sons) {
163                         std::reverse(T.begin(), T.end());
164                         G[x].push_back(T);
165                     }
166                     return std::vector<int> (1, x);
167                 } else {
168                     return sons[0];
169                 }
170             };
171
172             dfs(dfs, &t);
173         }
174     }
175     unique();
176 }
177
178 void show() {
179     unique();
180     add_tmp();
181     for (auto [l, r] : G) {
182         std::cout << mp[l] << " -> ";
183         int m = r.size();
184         for (auto &x : r) {
185             for (auto &y : x) {
186                 std::cout << mp[y] << " ";
187             }
188             if (&x != &(*r.rbegin())) {
189                 std::cout << "| ";
190             }
191         }
192         std::cout << "\n";
193     }
194 }
195
196 // 消去直接左递归
197 void remove_direct_left_recursion(int l) {
198     //remove left recursion
```

```
199         add_tmp();
200         auto r = G[l];
201         std::vector<std::vector<int>> A, B;
202         for (auto &x : r) {
203             if (x[0] == l) {
204                 A.push_back(x);
205             } else {
206                 B.push_back(x);
207             }
208         }
209         if (A.size() == 0) {
210             return;
211         }
212 
213         if (B.empty()) {
214             std::cerr << "Error: a -> a+\n";
215             exit(0);
216         }
217         int x = new_rule();
218         for (auto &y : B) {
219             y.push_back(x);
220         }
221         for (auto &y : A) {
222             y.erase(y.begin());
223             y.push_back(x);
224         }
225 
226         G[l] = B;
227         G[x] = A;
228         G[x].push_back(std::vector<int> (1, 0));
229         unique();
230     }
231 
232     // 消去间接左递归
233     void remove_left_recursion() {
234         //remove left recursion
235         add_tmp();
236 
237         std::vector<int> not_end;
238         for (auto &[l, r] : G) {
239             if (r.size() == 0) {
240                 continue;
241             }
242             not_end.push_back(l);
243         }
244         int m = not_end.size();
245 
246         for (int ii = 0; ii < m; ++ii) {
247             int i = not_end[ii];
248             for (int jj = 0; jj < ii; ++jj) {
```

```
249             int j = not_end[jj];
250             auto A = G[i], B = G[j];
251             for (auto y : A) {
252                 if (y[0] == j) {
253                     G[i].erase(std::find(G[i].begin(), G[i].end(), y));
254                     for (auto z : B) {
255                         std::vector<int> tmp = z;
256                         if (tmp.back() == 0) tmp.pop_back();
257                         tmp.insert(tmp.end(), y.begin() + 1, y.end());
258                         G[i].push_back(tmp);
259                     }
260                 }
261             }
262         }
263         remove_direct_left_recursion(i);
264     }
265     unique();
266 }
267
268 void get_first() {
269     for (int i = 0; i < n; ++i) {
270         if (!G.count(i) || G[i].size() == 0) {
271             first[i].insert(i);
272         }
273     }
274
275     std::vector<int> last_moify(n, 0);
276     bool modify = true;
277     for (int i = 0; modify; ++i) {
278         // std::cerr << "i = " << i << std::endl;
279         modify = false;
280         for (auto [l, r] : G) {
281             bool modify_l = false;
282             for (auto y : r) {
283                 bool empty = true;
284                 for (auto yi : y) {
285                     if (!empty) break;
286                     if (last_moify[yi] >= last_moify[l]) {
287                         for (auto x : first[yi]) {
288                             if (x == 0) continue;
289                             if (!first[l].count(x)) {
290                                 first[l].insert(x);
291                                 modify_l = true;
292                             }
293                         }
294                     }
295                     empty = first[yi].count(0);
296                 }
297                 if (empty) {
298                     if (!first[l].count(0)) {
```

```
                        first[l].insert(0);
                        modify_l = true;
                    }
                }
            }
            if (modify_l) {
                modify = true;
                last_moify[l] = i;
            }
        }
    }
}

void get_follow() {
    get_first();

    std::set<int> end;

    for (int i = 0; i < n; ++i) {
        if (!G.count(i) || G[i].size() == 0) {
            end.insert(i);
        }
    }

    // -1 means $
    follow[start].insert(-1);

    bool modify = true;
    for (int i = 0; modify; ++i) {
        modify = false;
        for (auto [l, r] : G) {
            for (auto y : r) {
                for (int j = 0; j < y.size(); ++j) {
                    int yi = y[j];
                    if (end.count(yi))continue;
                    bool empty = true;
                    for (int k = j + 1; k < y.size(); ++k) {
                        int yk = y[k];
                        if (!empty) break;
                        for (auto x : first[yk]) {
                            if (x == 0) continue;
                            if (!follow[yi].count(x)) {
                                follow[yi].insert(x);
                                modify = true;
                            }
                        }
                        empty = first[yk].count(0);
                    }
                    if (empty) {
                        for (auto x : follow[l]) {
```

```
                            if (!follow[yi].count(x)) {
                                follow[yi].insert(x);
                                modify = true;
                            }
                        }
                    }
                }
            }
        }
    }

    void get_select() {
        for (auto [l, r] : G) {
            int m = r.size();
            select[l].resize(m);
            for (int i = 0; i < m; ++i) {
                auto &y = r[i];
                bool empty = true;
                for (auto yi : y) {
                    if (!empty) break;
                    for (auto x : first[yi]) {
                        if (x == 0) continue;
                        select[l][i].insert(x);
                    }
                    empty = first[yi].count(0);
                }
                if (empty) {
                    for (auto x : follow[l]) {
                        select[l][i].insert(x);
                    }
                }
            }
        }
    }

    bool check_ll1() {
        for (auto [l, r] : G) {
            int m = r.size();
            for (int i = 0; i < m; ++i) {
                for (int j = i + 1; j < m; ++j) {
                    if ((select[l][i] & select[l][j]).size()) {
                        return false;
                    }
                }
            }
        }
        return true;
    }

```

```
399     void put_graph() {
400         std::vector<int> not_end, end;
401 
402         for (int i = -1; i < n; ++i) {
403             if (!G.count(i) || G[i].size() == 0) {
404                 end.push_back(i);
405             } else {
406                 not_end.push_back(i);
407             }
408         }
409 
410         std::vector<std::vector<int>> biao(not_end.size(), std::vector<int> (end.size(), -1));
411         for (auto [l, r] : G) {
412             int m = r.size();
413             for (int i = 0; i < m; ++i) {
414                 for (auto x : select[l][i]) {
415                     biao[std::find(not_end.begin(), not_end.end(), l) - not_end.begin()][std::find(
                            end.begin(), end.end(), x) - end.begin()] = i;
416                 }
417             }
418         }
419 
420         // 用md格式打印表
421         std::cout << "| | ";
422         for (auto x : end) {
423             std::cout << (x == -1 ? "$" : mp[x]) << " | ";
424         }
425         std::cout << "\n";
426 
427         std::cout << "| --- |";
428         for (auto x : end) {
429             std::cout << " --- |";
430         }
431         std::cout << "\n";
432 
433         for (int i = 0; i < not_end.size(); ++i) {
434             std::cout << "| " << mp[not_end[i]] << " | ";
435             for (int j = 0; j < end.size(); ++j) {
436                 if (biao[i][j] == -1) {
437                     std::cout << " | ";
438                 } else {
439                     std::cout << mp[not_end[i]] << " -> ";
440                     for (auto x : G[not_end[i]][biao[i][j]]) {
441                         std::cout << mp[x] << " ";
442                     }
443                     std::cout << " | ";
444                 }
445             }
446             std::cout << "\n";
447         }
```

```
448     }
449
450 };
451
452 gramma_token parse(std::vector<rule> rules) {
453     std::vector<std::string> mp;
454     std::map<std::string, int> id;
455     std::string epislon = "ε";
456
457     int n = rules.size();
458     std::vector<rule_token> tokens(n);
459
460     id[epislon] = 0;
461     mp.push_back(epislon);
462
463     auto get_id = [&] (std::string x) {
464         if (!id.count(x)) {
465             id[x] = id.size();
466             mp.push_back(x);
467         }
468         return id[x];
469     };
470     for (int i = 0; i < n; ++i) {
471         tokens[i].L = get_id(rules[i].L);
472         tokens[i].R.resize(rules[i].R.size());
473     }
474
475     for (int i = 0; i < n; ++i) {
476         int m = rules[i].R.size();
477         for (int j = 0; j < m; ++j) {
478             auto r = rules[i].R[j];
479             while (r.length() > 0) {
480                 bool modify = false;
481                 // std::cerr << "!" << r << std::endl;
482                 for (auto [str, ID] : id) {
483                     if (r.length() < str.length()) {
484                         continue;
485                     } else {
486                         // std::cerr << str << " " << r.substr(0, str.length()) << "\n";
487                         if (r.substr(0, str.length()) == str) {
488                             r = r.substr(str.length());
489                             tokens[i].R[j].push_back(get_id(str));
490                             modify = true;
491                             break;
492                         }
493                     }
494                 }
495                 if (!modify) {
496                     auto tmp = r.substr(0, 1);
497                     r = r.substr(1);
```

```
498                     tokens[i].R[j].push_back(get_id(tmp));
499                 }
500             }
501         }
502     }
503 
504     gramma_token g;
505     g.mp = mp;
506 
507     for (auto x : tokens) {
508         for (auto y : x.R) {
509             g.G[x.L].push_back(y);
510         }
511     }
512 
513     g.n = g.mp.size();
514 
515     return g;
516 }
517 
518 
519 int main() {
520     std::vector<rule> g;
521     std::string line;
522     while (std::getline(std::cin, line)) {
523         g.push_back(rule(line));
524     }
525 
526 
527 
528     auto g_token = parse(g);
529     g_token.show();
530 
531     // std::cout << "After remove left recursion:\n";
532     std::cout << "\n";
533 
534     g_token.remove_left_recursion();
535     g_token.remove_rep();
536 
537     std::cout << "After remove left recursion:\n";
538     g_token.show();
539 
540     g_token.get_first();
541 
542     std::cout << "\n";
543 
544     std::cout << "First:\n";
545 
546     for (int i = 0; i < g_token.n; ++i) {
547         std::cout << g_token.mp[i] << ": ";
```

```
548         for (auto x : g_token.first[i]) {
549             std::cout << g_token.mp[x] << " ";
550         }
551         std::cout << std::endl;
552     }
553 
554     g_token.get_follow();
555 
556     std::cout << "\n";
557     std::cout << "FOLLOW:\n";
558     for (int i = 0; i < g_token.n; ++i) {
559         std::cout << g_token.mp[i] << ": ";
560         for (auto x : g_token.follow[i]) {
561             std::cout << (x == -1 ? "$" : g_token.mp[x]) << " ";
562         }
563         std::cout << std::endl;
564     }
565 
566     g_token.get_select();
567     std::cout << "\n";
568     std::cout << "SELECT:\n";
569     for (auto [l, r] : g_token.G) {
570         int m = r.size();
571         for (int i = 0; i < m; ++i) {
572             std::cout << g_token.mp[l] << " -> ";
573             for (auto x : r[i]) {
574                 std::cout << g_token.mp[x] << " ";
575             }
576             std::cout << ": ";
577             for (auto x : g_token.select[l][i]) {
578                 std::cout << (x == -1 ? "$" : g_token.mp[x]) << " ";
579             }
580             std::cout << std::endl;
581         }
582     }
583 
584     std::cout << (g_token.check_ll1() ? "LL1" : "Not LL1") << std::endl;
585     std::cout << "\n";
586     if (g_token.check_ll1()) {
587         g_token.put_graph();
588     }
589     return 0;
590 }
```