中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle

编 译 原 理

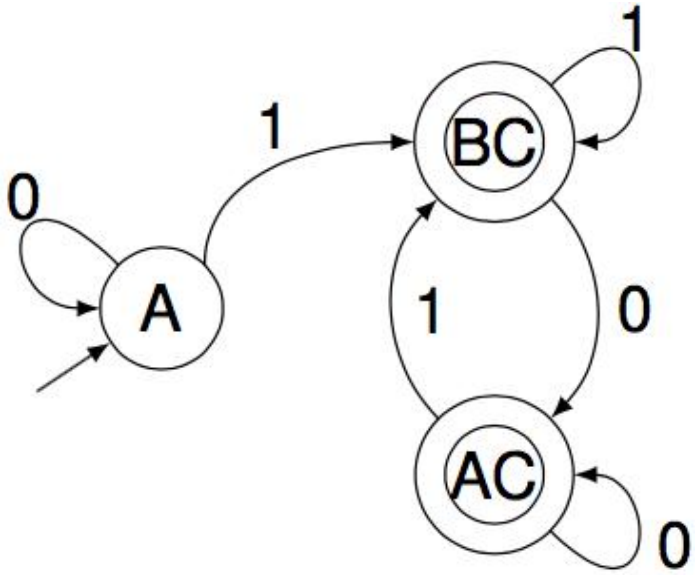
第三章 词法分析（3）

郑馥丹

zhengfd5@mail.sysu.edu.cn

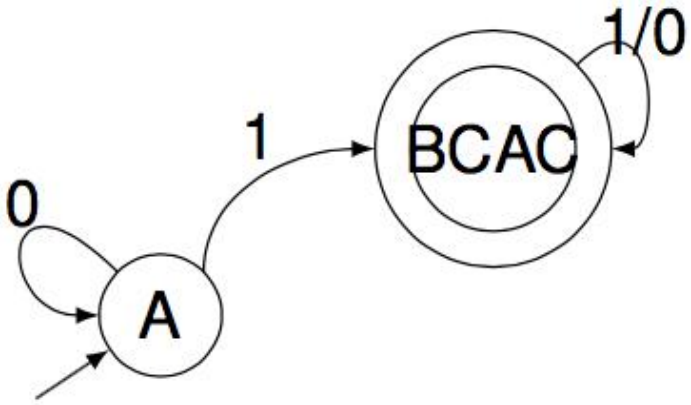
4. DFA化简[Reduction]

- 任务：**去掉多余状态，合并等价状态**
  - 多余状态：从开始状态出发无法到达的状态。
  - 等价状态：两个状态s和t等价的条件是:
    - ✓ 1. **一致性条件**—状态s和t必须同为可接受状态或不可接受状态。
    - ✓ 2. **蔓延性条件**—对于所有输入符号，状态s和状态t必须转换到等价的状态里。



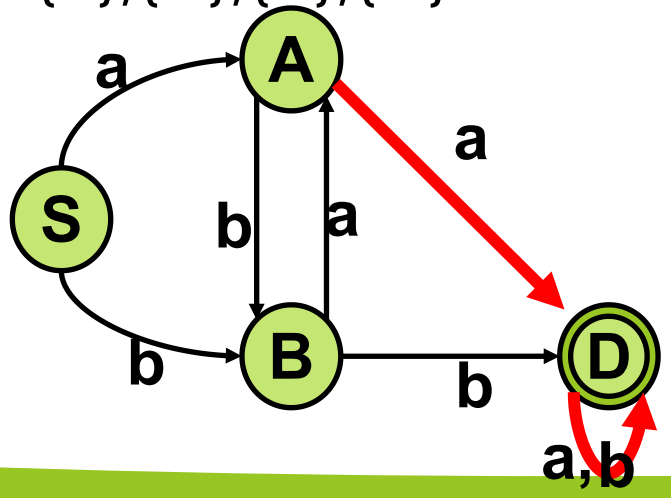
1. BC和AC**同为可接受状态**
2. BC和AC对所有输入符号**都转移到等价的状态里**
   - BC on '0' → AC, AC on '0' → AC
   - BC on '1' → BC, AC on '1' → BC

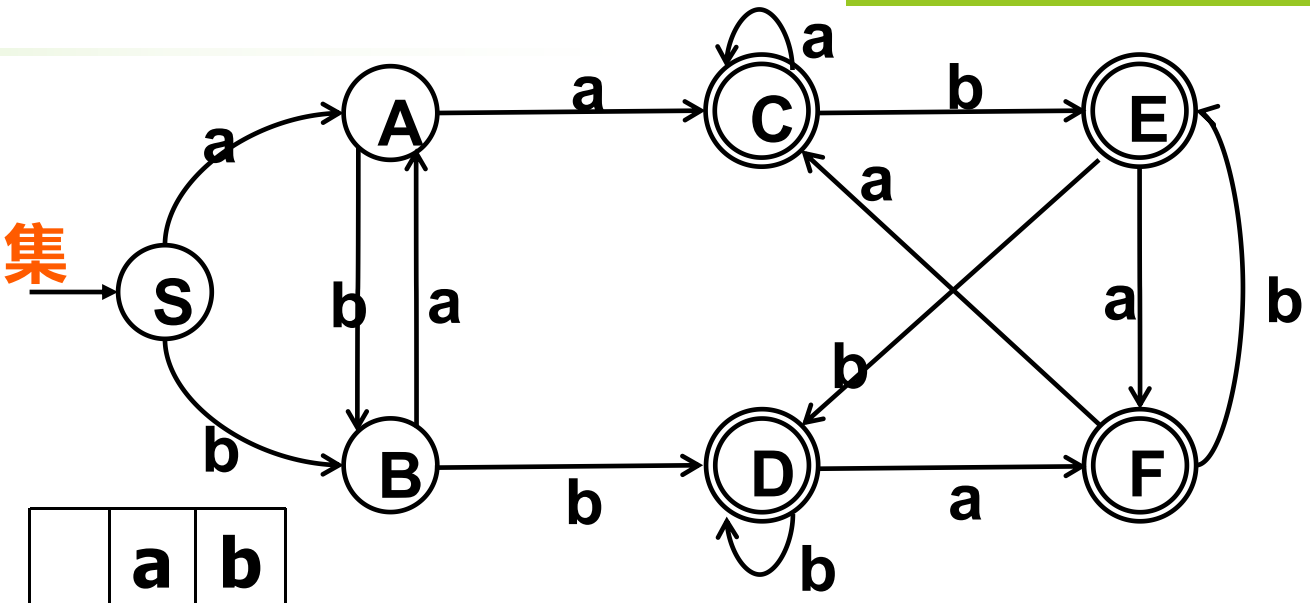
因此：BC和AC是**等价状态，可合并**。



4. DFA化简[Reduction]

- 例1：将右图DFA化简
  - **Step 1：将状态分成非终态和终态集**
    - ✓ **{S,A,B} {C,D,E,F}**
  - **Step 2：寻找子集中不等价状态**
    - ✓ {S,A,B}=>{S}{A,B}=>{S}{A}{B}
    - ✓ {C,D,E,F}
  - **Step 3：令D代表{C,D,E,F}**
    - ✓ {S},{A},{B},{D}





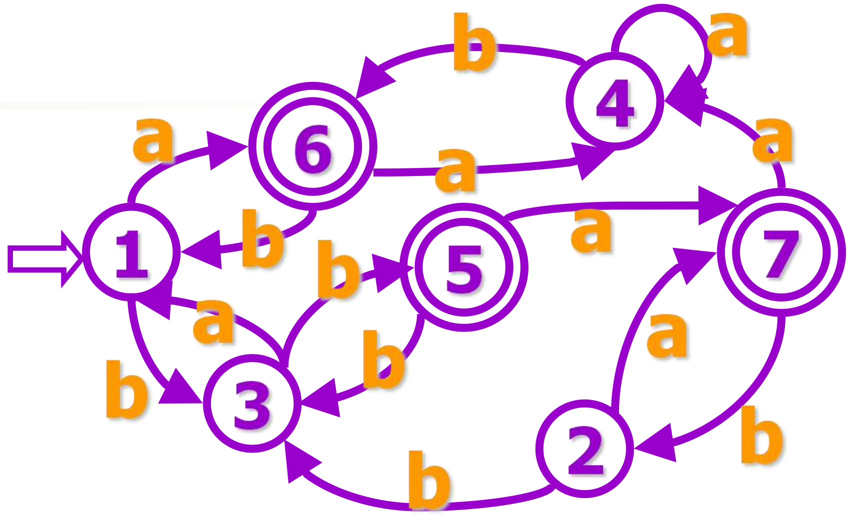
	a	b
S	A	B
A	C	B
B	A	D
C	C	E
D	F	D
E	F	D
F	C	E

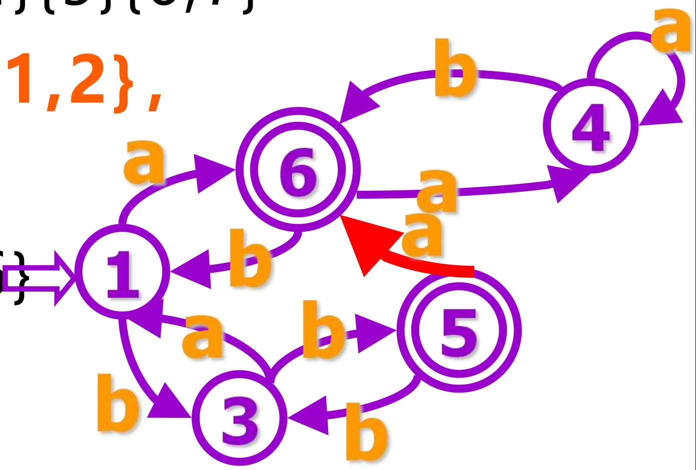
	a	b
S	A	B
A	C	B
B	A	D
CF	C	E
DE	F	D

	a	b
S	A	B
A	C	B
B	A	D
CFDE	CF	DE

4. DFA化简[Reduction]

- 例2：将右图DFA化简
  - Step 1：将状态分成非终态和终态集
    - ✓ {1,2,3,4} {5,6,7}
  - Step 2：寻找子集中不等价状态
    - ✓ {1,2,3,4}=>{1,2}{3,4}
    - ✓ {3,4}=>{3}{4}
    - ✓ {5,6,7}=>{5}{6,7}
    - ✓ 最终：{1,2}{3}{4}{5}{6,7}
  - Step 3：令1代表{1,2}，6代表{6,7}
    - ✓ {1},{3},{4},{5},{6}





	a	b
1	6	3
2	7	3
3	1	5
4	4	6
5	7	3
6	4	1
7	4	2

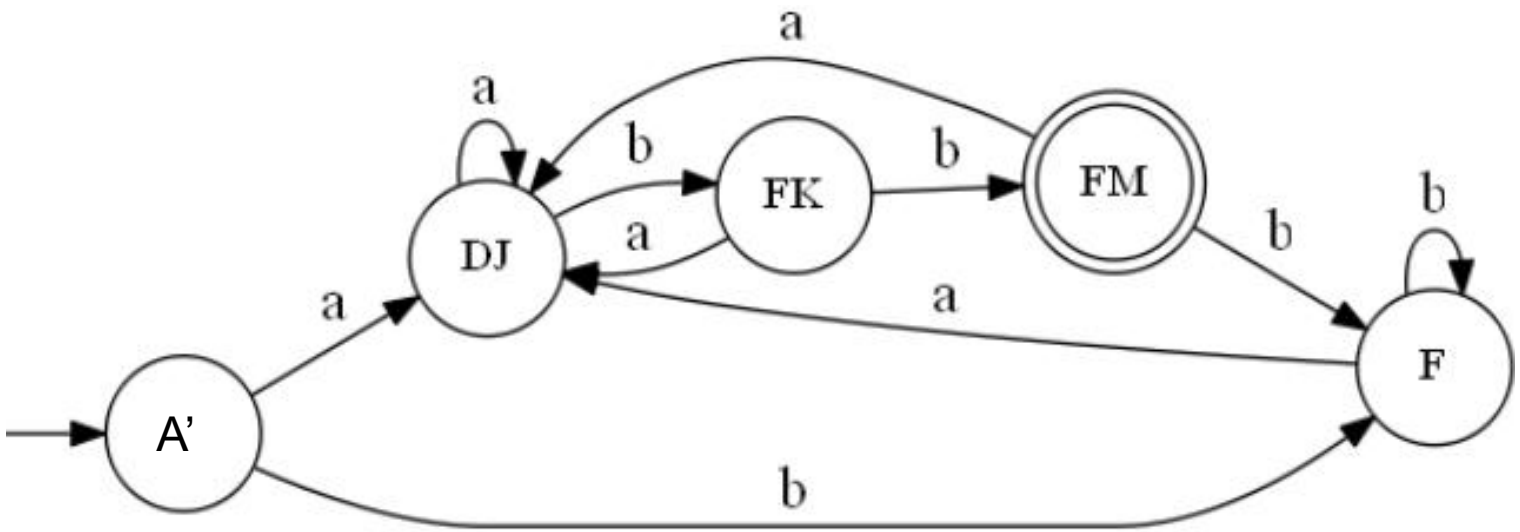
	a	b
1	6	3
2	7	3
3	1	5
4	4	6
5	7	3
6	4	1
7	4	2

	a	b
1	6	3
2	7	3
3	1	5
4	4	6
5	7	3
6	4	1
7	4	2

	a	b
1	6	3
2	7	3
3	1	5
4	4	6
5	7	3
6	4	1
7	4	2

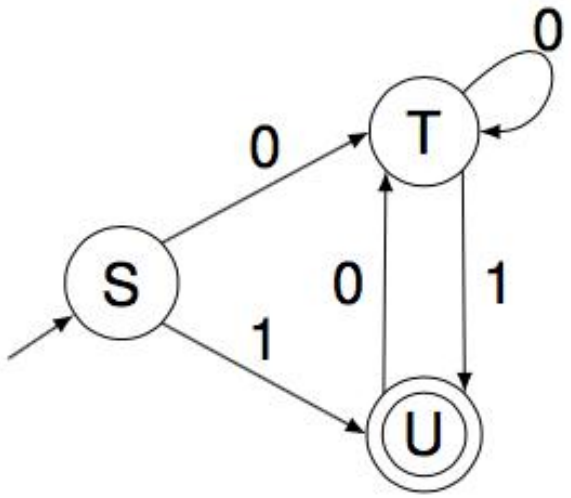
随堂练习(5)

- 将下图DFA化简



5. DFA→Table-drive Implementation

- DFA可转成Table-drive Code



alphabet

state

	0	1
S	**T**	**U**
T	**T**	**U**
U	**T**	**x**

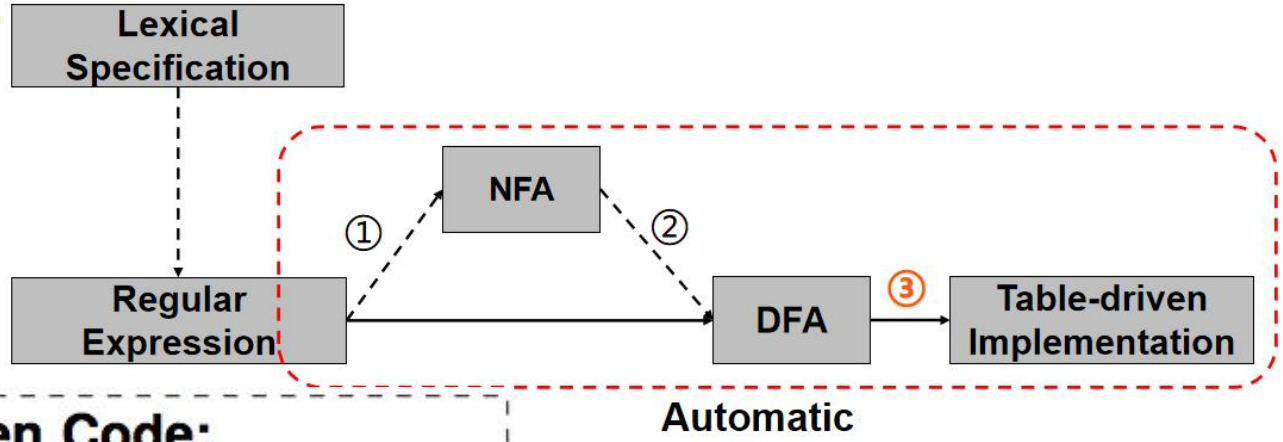


Table-driven Code:

```
DFA() {
    state = "S";
    while (!done) {
        ch = fetch_input();
        state = Table[state][ch];
        if (state == "x")
            print("reject");
    }
    if (state ∈ F)
        printf("accept");
    else
        printf("reject");
}
```

Q: which is/are accepted?

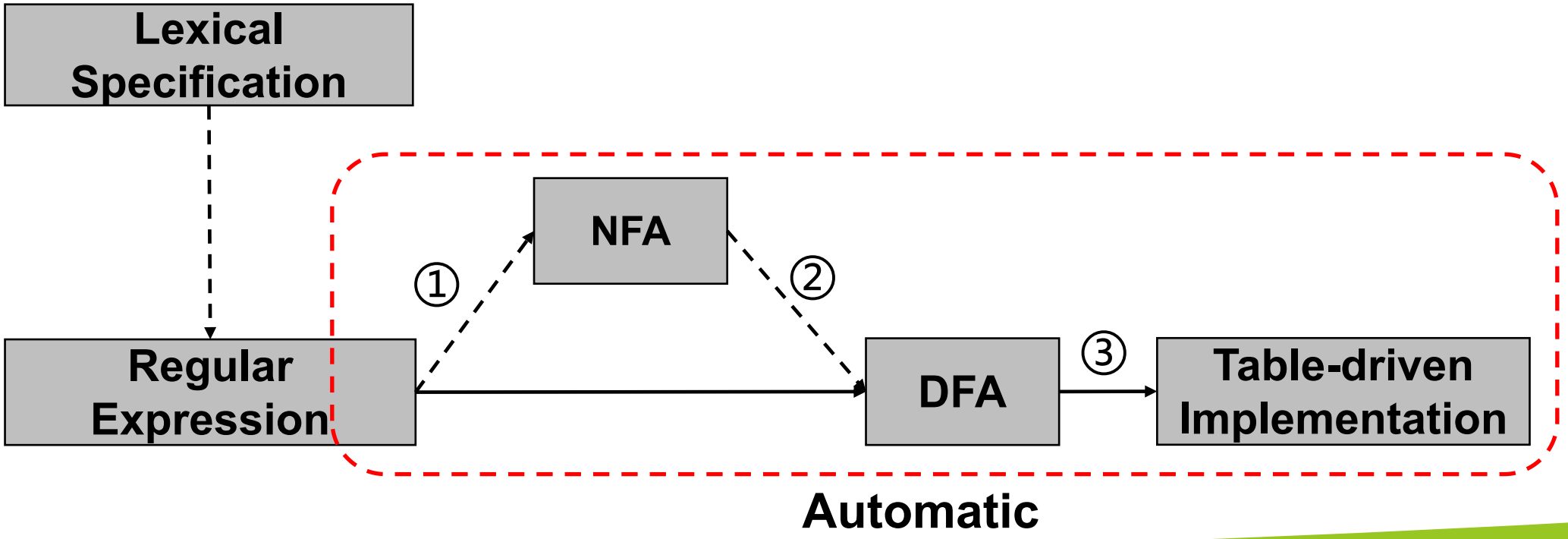
111

000

001

转换流程[Conversion Flow]

- 流程：RE→NFA→DFA→Table-drive Implementation
  - ① RE→NFA
  - ② NFA→DFA
  - ③ DFA→Table-drive Implementation



6. 一些讨论

- 关于Table-drive Implementation
  - 表格是一种高效实现[efficient]
    - ✓ 仅需**有穷空间O(S x ∑)**
      - 转换表的Size
    - ✓ 仅需**有穷时间O(input length)**
      - 状态转换的个数
  - 表格实现的优劣
    - ✓ pros：在给定的状态和输入下能轻松找到转换
    - ✓ cons：当输入字母很大时，会占用大量空间，但大多数状态对大多数输入符号没有任何移动

6. 一些讨论

- 关于**空间复杂度[Space Complexity]**
  - NFA在任何时刻可能有多个状态
  - DFA：
    - ✓ 如果NFA有N个状态，则DFA一定在这N个状态的某个子集中
    - ✓ 非空子集：2^N-1
    - ✓ 空间复杂度：$O(2^N)$，其中N为NFA中的原始状态个数

6. 一些讨论

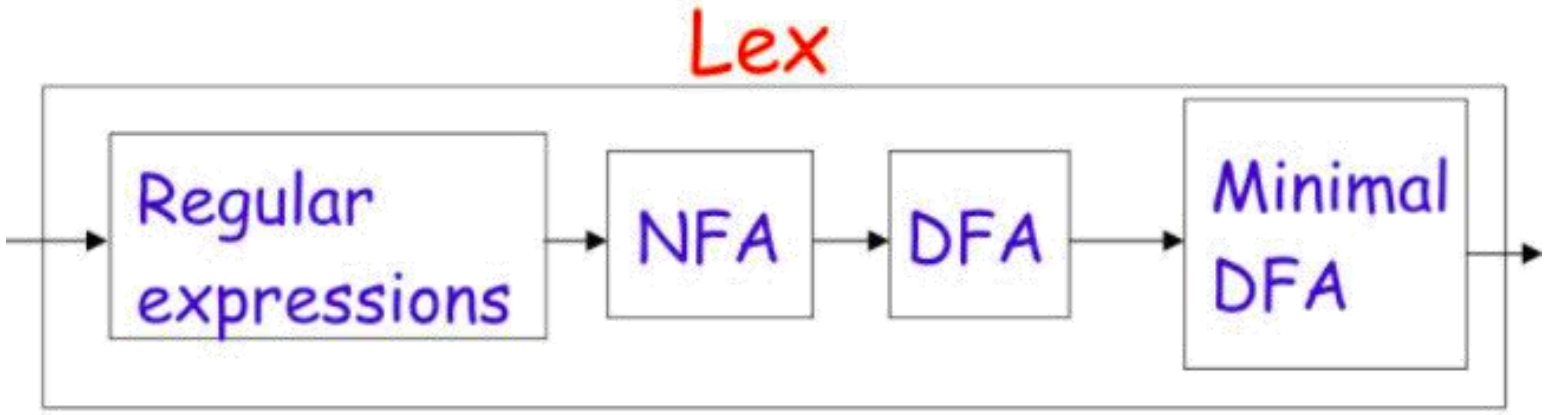
- 关于**时间复杂度[Time Complexity]**
  - DFA
    - ✓ 需要**O(|X|)步**，其中|X|是输入长度
    - ✓ **每一步花费O(1)**常数时间
      - 若当前状态是S且输入为c，则读表T[S,c]，更新当前状态为T[S,c]
    - ✓ 时间复杂度为**O(|X|)**
  - NFA
    - ✓ 需要**O(|X|)步**，其中|X|是输入长度
    - ✓ **每一步花费O(N^2)**时间，其中N为状态的个数
      - 当前状态是一组潜在状态，最多N个
      - 对于输入c，必须合并所有T[Spotential, c]，最多N次，每次合并操作花费O(N)时间
    - ✓ 时间复杂度为**O(|X|*N^2)**

CONTENTS

目 录

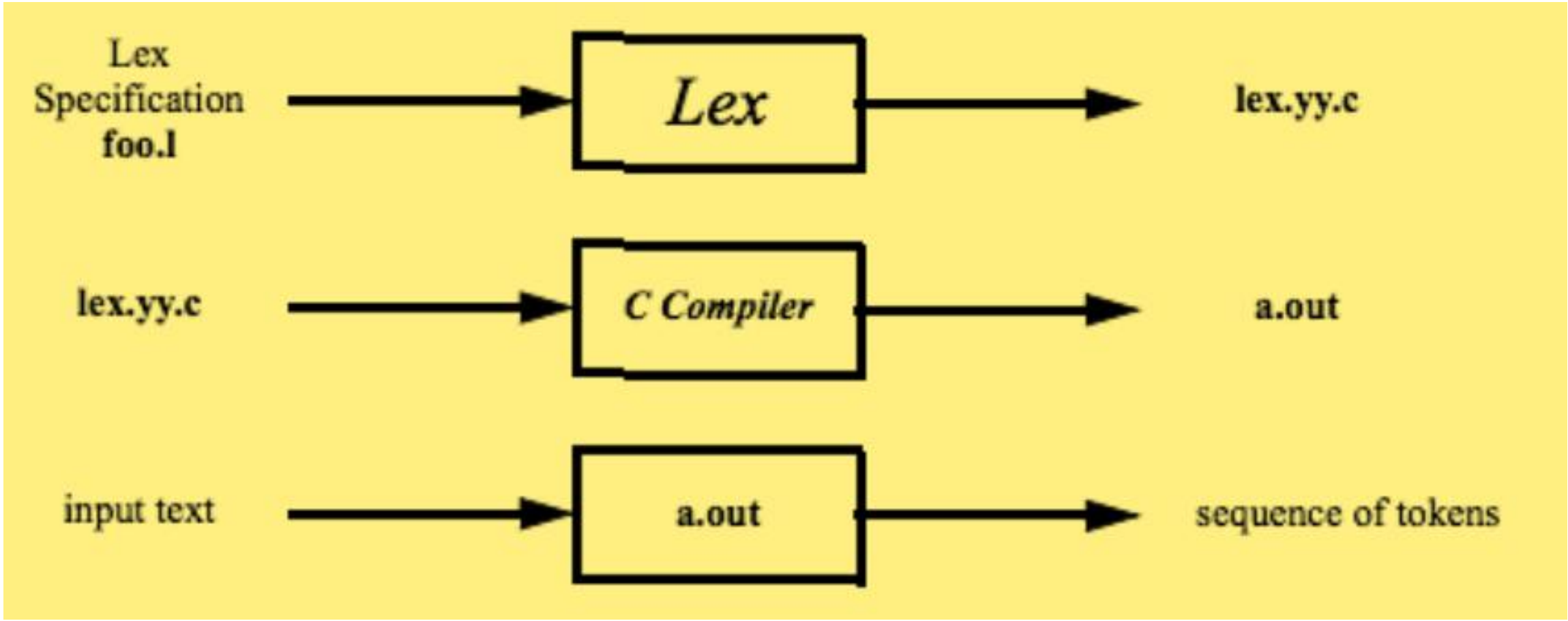
1. 实际实现

- Lex: RE → NFA → DFA → Table



- 大多数其他自动词法分析器也选择DFA而不是NFA
  - 用空间换取速度[Trade off space for speed]

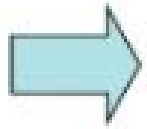
2. Lex

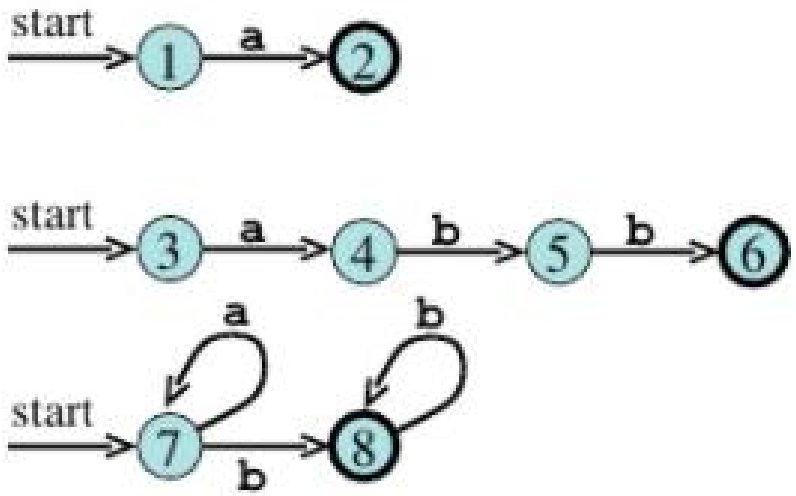

Lex
Specification
foo.l
Lex
lex.yy.c
lex.yy.c
C Compiler
a.out
input text
a.out
sequence of tokens

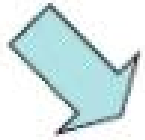
2. Lex

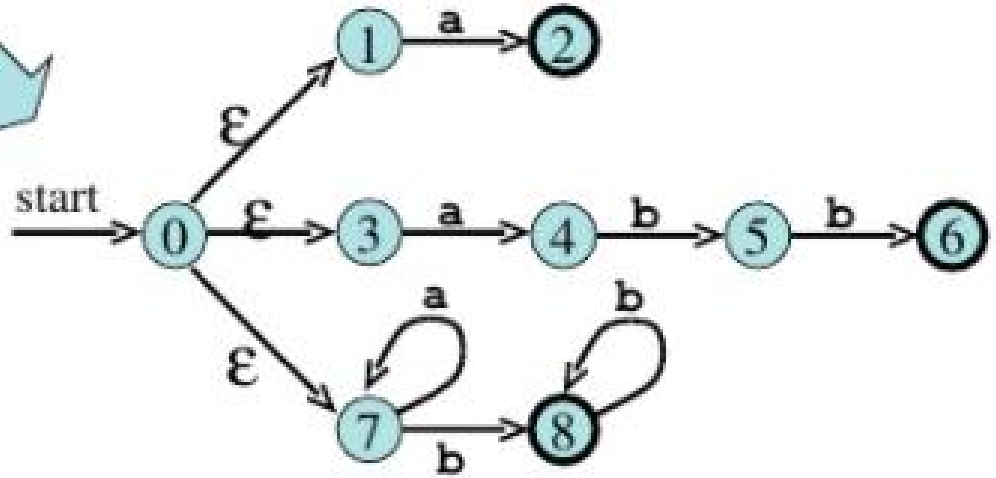
- 例：假设现有3种模式，对应3个NFA
  - 将3个NFA组合成1个NFA
  - 添加开始状态和ε转移

a	{ $action_1$ }
abb	{ $action_2$ }
a*b+	{ $action_3$ }









2. Lex

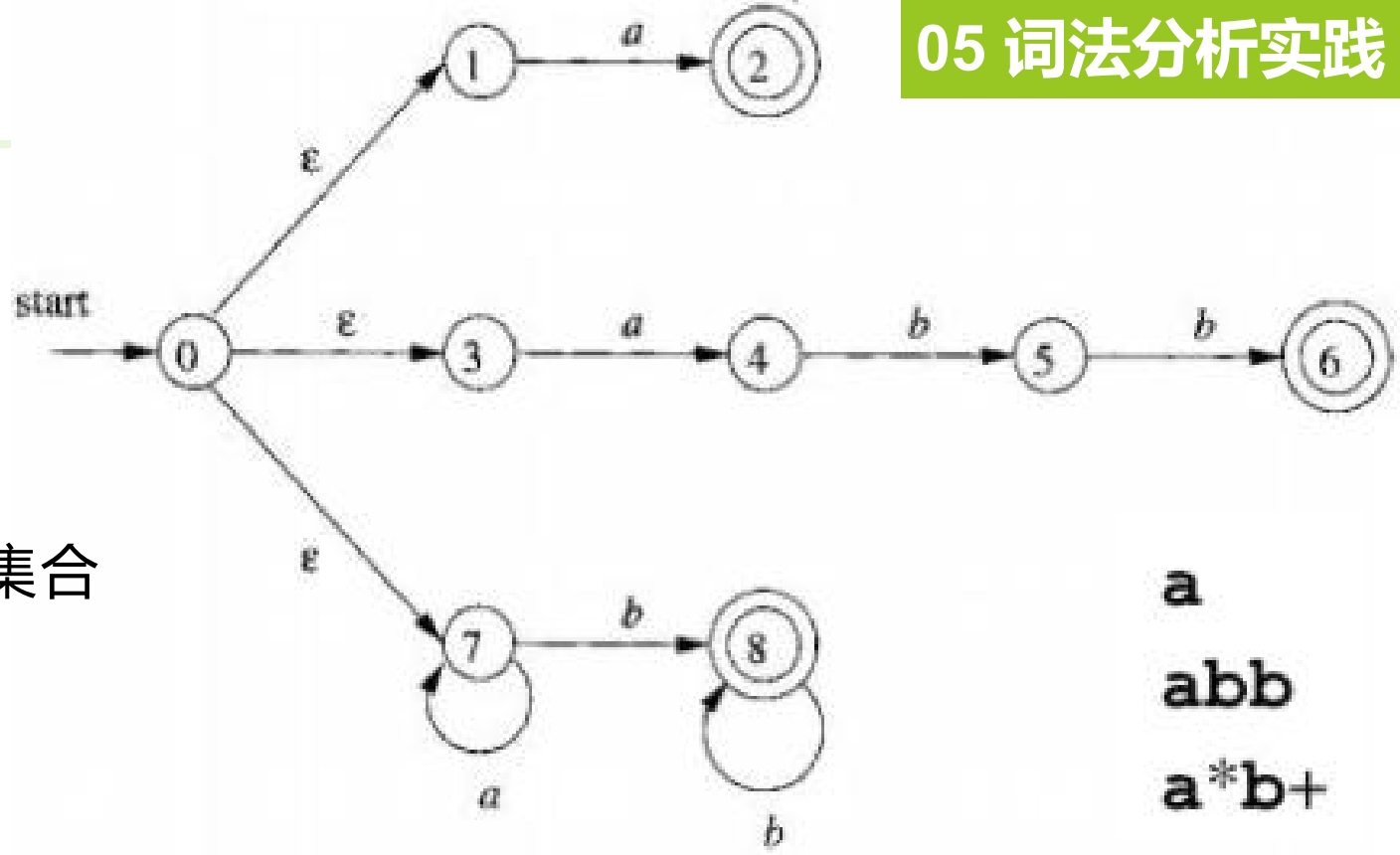
```
ptn1    a
ptn2    abb
ptn3    a*b+

%%

{ptn1}  { printf("\n<%s, %s>", "ptn1", yytext); }
{ptn2}  { printf("\n<%s, %s>", "ptn2", yytext); }
{ptn3}  { printf("\n<%s, %s>", "ptn3", yytext); }

%%

int main(){
  yylex();
  return 0;
}
```

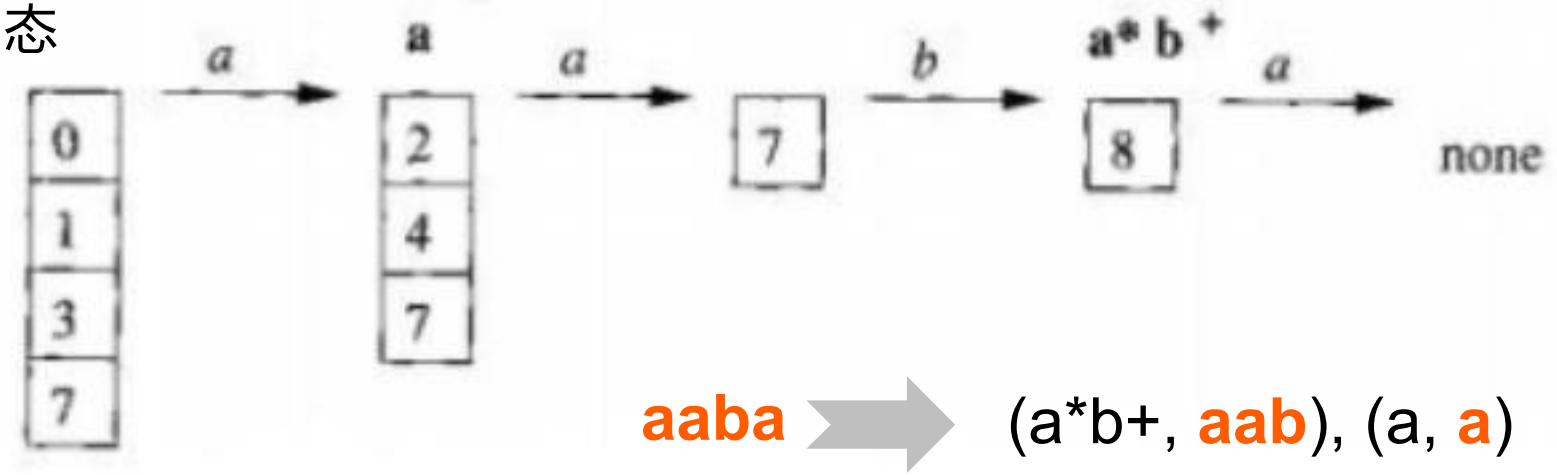
```
$flex lex.l
$clang lex.yy.c -o mylex -ll
```

```
root@aa51dde06c76:~/test# echo "aaba" | ./mylex

<ptn3, aab>
<ptn1, a>
root@aa51dde06c76:~/test# echo "abba" | ./mylex

<ptn2, abb>
<ptn1, a>
```

2. Lex

- NFA
  - 输入：**aaba**
    - ✓ ε-closure(0) = {0, 1, 3, 7}
    - ✓ 寻找一组**包含接受状态**的状态集合
    - ✓ 状态8：a*b+匹配成功
      - aab即为词素
      - 继续寻找接受状态

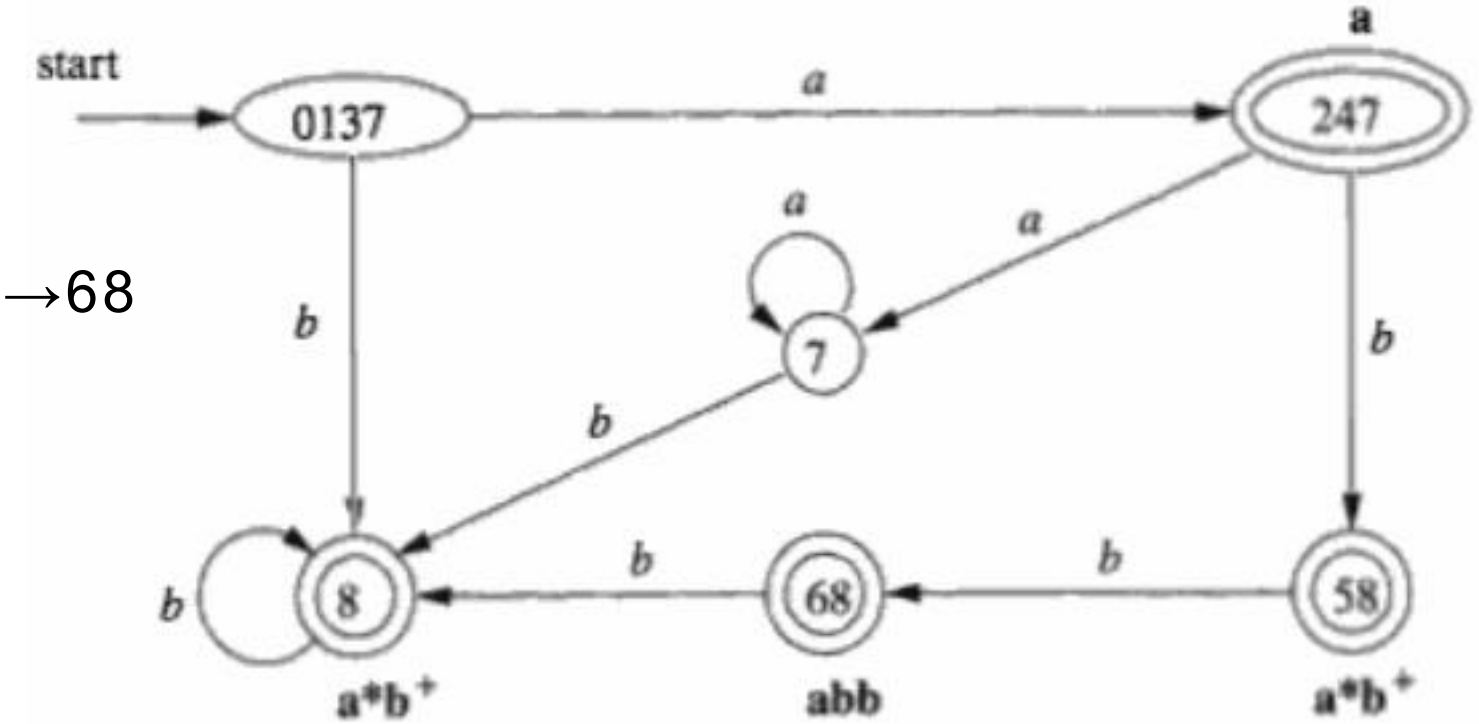




aaba ⇒ (a*b+, aab), (a, a)

2. Lex

- DFA
  - 输入：**abba**
    - ✓ 状态序列：0137→247→58→68
    - ✓ 状态68：abb匹配成功
      - abb即为词素
      - 继续寻找接受状态



a

abb

a*b+

2. Lex

```
a       { action1 }
abb     { action2 }
a*b+    { action3 }
```

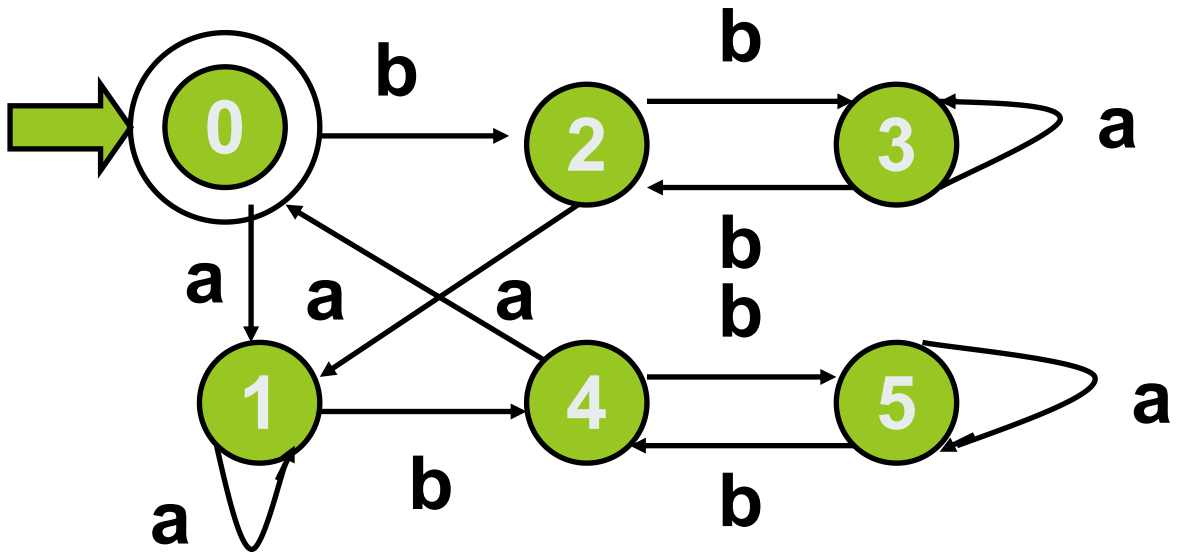
- 若有多种匹配的可能性?
  - **寻找最长匹配**
    - ✓ 例：输入**aabbb**，符合pattern2和pattern3，取最长匹配——pattern3
  - **先出现的规则优先级更高**
    - ✓ 例：输入**abb**，符合pattern2和pattern3，取先出现的pattern2

2. Lex

- 如何匹配关键字[keywords]?
  - 方法1：为关键字创建正则表达式，并将它们放在标识符的正则表达式之前，让他们具有更高的优先级
    - ✓ 会导致更臃肿的FA
  - 方法2：使用相同的正则表达式识别关键字和标识符，但使用特殊的**关键字表[keyword table]**进行区分
    - ✓ 会导致更精简的FA
    - ✓ 但是需要额外的表查找
  - **方法2更常用**

第三章课后作业

- 将下图DFA最小化



第三章课后作业

- 提交要求：
  - 文件命名：学号-姓名-第三章作业；
  - 文件格式：.pdf文件；
  - 手写版、电子版均可；若为手写版，则拍照后转成pdf提交，但**须注意将照片旋转为正常角度，且去除照片中的多余信息**；电子版如word等转成pdf提交；
  - 提交到超算习堂（第三章作业）处；
  - 提交ddl：**3月20日晚上12:00**；
  - **重要提示：不得抄袭！**