编译原理
LR 语法分析器实验报告

编译环境： Microsoft Visual Studio 2019

Windows SDK **版本：** 10.0

平台工具集： Visual Studio 2019（v142）

语言： C++

1、需求：LR 语法分析程序的设计与实现。

程序设计 2

题目：语法分析程序的设计与实现。

实验内容：编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生。

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow (E) \mid \text{num}$

实验要求：在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

方法 1：编写递归调用程序实现自顶向下的分析。

方法 2：编写 LL(1)语法分析程序，要求如下。

(1) 编程实现算法 4.2，为给定文法自动构造预测分析表。

(2) 编程实现算法 4.1，构造 LL(1)预测分析程序。

方法 3：编写语法分析程序实现自底向上的分析，要求如下。

(1) 构造识别该文法所有活前缀的 DFA。

(2) 构造该文法的 LR 分析表。

(3) 编程实现算法 4.3，构造 LR 分析程序。

2、核心算法：

PS:除题目的基础要求外，还额外实现了对任意 SLR1 文法，自动生成拓广文法，且求该文法的 first 集和 follow 集，并且自动生成 LR0 项目集规范簇和识别该文法所有活前缀的 DFA，然后根据该 DFA 自动生成 SLR1 分析表。

手动分析：

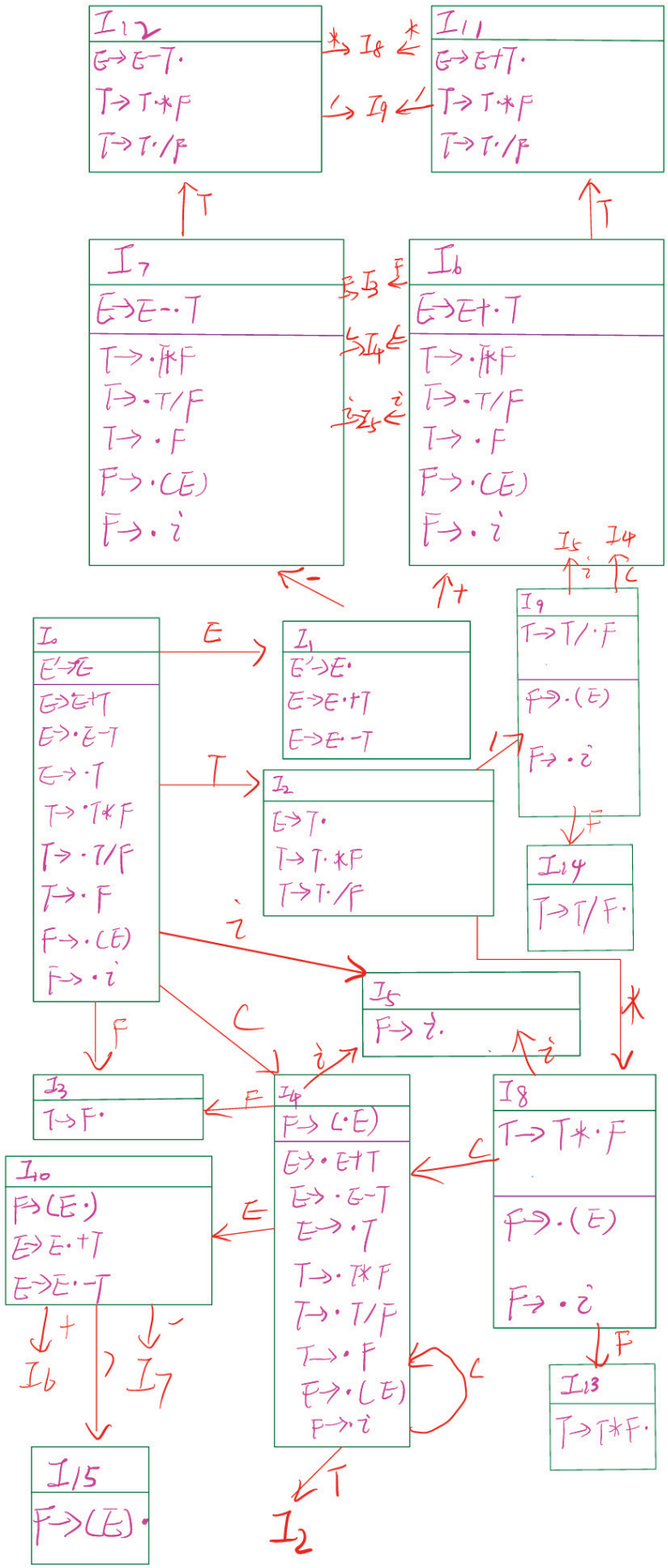
①首先对文法进行拓广，求 first 集和 follow 集：

0. E'→E
1. E→E+T
2. E→E-T
3. E→T
4. T→T*F
5. T→T/F
6. T→F
7. F→(E)
8. F→i

	FIRST	FOLLOW
E'	(, i	$
E	(, i	$, +, -,)
T	(, i	$, +, -, *, /,)
F	(, i	$, +, -, *, /,)

发现该文法是一个 SLR1 文法。

②所以构造 LR0 项目集规范簇及识别该文法所有活前缀的 DFA：


I12
E→E-T·
T→T·*F
T→T·/F
I11
E→E+T·
T→T·*F
T→T·/F
I8
I9
*
/
T
I7
E→E-·T
T→·T*F
T→·T/F
T→·F
F→·(E)
F→·i
I6
E→E+·T
T→·T*F
T→·T/F
T→·F
F→·(E)
F→·i
E
I3
+
I4
-
i
I5
I0
E→·E
E→·E+T
E→·E-T
E→·T
T→·T*F
T→·T/F
T→·F
F→·(E)
F→·i
I1
E'→E·
E→E·+T
E→E·-T
I2
E→T·
T→T·*F
T→T·/F
I9
T→T/·F
F→·(E)
F→·i
I14
T→T/F·
I5
F→i·
I3
T→F·
I4
F→(·E)
E→·E+T
E→·E-T
E→·T
T→·T*F
T→·T/F
T→·F
F→·(E)
F→·i
I8
T→T*·F
F→·(E)
F→·i
I10
F→(E·)
E→E·+T
E→E·-T
I6
I7
I15
F→(E)·
I13
T→T*F·
I2

③构造识别该文法的 SLR0 分析表：

状态	action								goto		
	+	−	*	/	(	)	i	$	E	T	F
0					S4		S5		1	2	3
1	S6	S7						ACC			
2	r3	r3	S8	S9		r3		r3			
3	r6	r6	r6	r6		r6		r6			
4					S4		S5		10	2	3
5	r8	r8	r8	r8		r8		r8			
6					S4		S5			11	3
7					S4		S5			12	3
8					S4		S5				13
9					S4		S5				14
10	S6	S7				S15					
11	r1	r1	S8	S9		r1		r1			
12	r2	r2	S8	S9		r2		r2			
13	r4	r4	r4	r4		r4		r4			
14	r5	r5	r5	r5		r5		r5			
15	r7	r7	r7	r7		r7		r7			

算法结构分析：

④整个程序的模型如图所示：

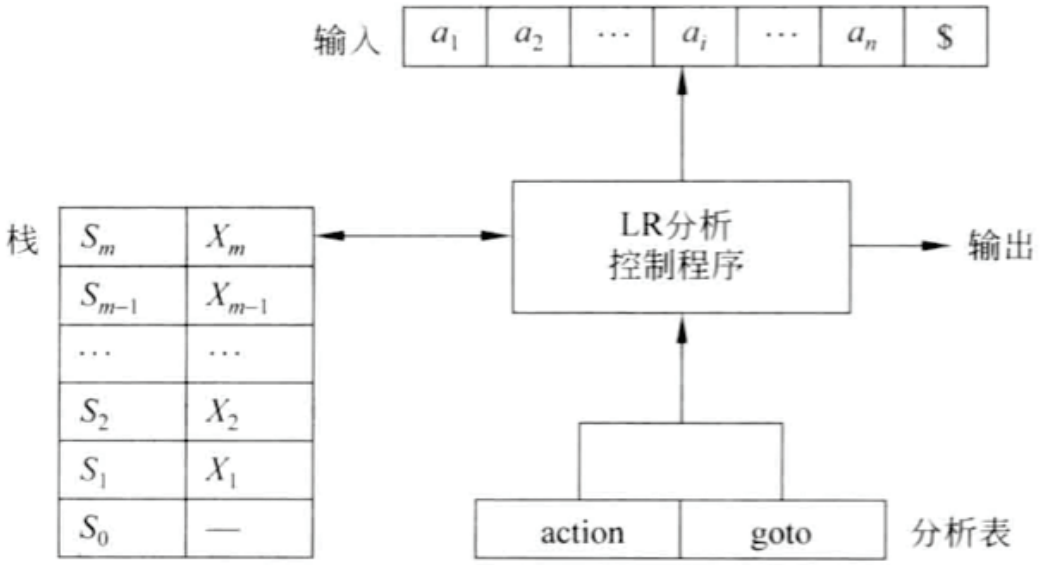


图 4-12　LR 分析程序的模型

状态栈、符号栈和分析表的数据结构详见【3、变量和函数】。

⑤预测分析控制程序算法是利用课本上的算法：

算法 4.3　LR 分析程序。

输入：文法 G 的一张分析表和一个输入符号串 ω。

输出：若 $\omega \in L(G)$，得到 ω 的自底向上的分析，否则报错。

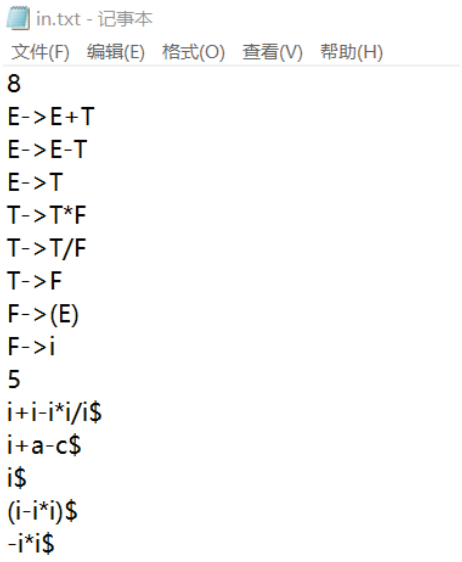
方法：

首先初始化，将初始状态 S_0 入栈，将 $\omega\$$ 存入输入缓冲区中；并置 ip 指向 $\omega\$$ 的第一个符号

```
do {
    令 S 是栈顶状态，a 是 ip 所指向的符号；
    if(action[S,a]=shift S′) {
        把 a 和 S′分别压入符号栈和状态栈的栈顶；
        推进 ip，使它指向下一个输入符号；
    };
    else if (action[S,a]=reduce by A→β) {
            从栈顶弹出|β|个符号；
            令 S′是当前栈顶状态，把 A 和 goto[S′,A]分别压入符号栈和状态栈的栈顶；
            输出产生式 A→β；
        };
        else if (action[S,a]=accept) return;
              else error();
} while(1);
```

输入形式：

文件输入：



格式为：
产生式个数n
{
    ……
    第i(1<=i<=n)个产生式（例如A->a，按照输入顺序分别从1到n编号，拓广产生式为0号，规定终结符和非终结符都是一个字符且非终结符都为大写字母）
    ……
}
要分析的字符串个数n
{
    ……
    第i(1<=i<=n)个要分析的字符串（要求以$结尾）
    ……
}

输出形式：

先输出该文法的拓广文法，再输出该拓广文法的first集和follow集，再输出SLR0分析表，然后输出用户输入的待分析字符串的分析过程，
详见【5、测试样例】所示：

3、变量和函数：

类和枚举：

```
enum RS//动作的枚举
{
    shift,//移进
```

```
    reduce,//规约
    accept//接受
};
//假定非终结符仅为一个大写字母，终结符仅为一个小写字母和各种符号，且用~代替epsilon
class PF//Production formula,产生式的类
{
public:
    char left;//产生式的左部
    string right;//产生式的右部
    PF(char l, string r)//构造函数
    {
        left = l;
        right = r;
    }
};
class ITEM//项目集
{
public:
    ITEM();//默认构造函数
    ITEM(int NO);//构造函数
    int NO;//项目集编号
    set<pair<int, int>> production;//项目，第一个int代表产生式编号，第二个int代表.的位置
    set<pair<char, int>> go_to;//该项目集指向的项目集，第一个char代表箭头上的字符，第二个int代表指向的项目集编号
    bool operator<(const ITEM& item)const;//重载<操作符，以便使用set
};
class action//动作类
{
public:
    action(RS rs,int num);//重写构造函数
    string ret_action();//返回该动作
    action();//初始
    RS rs;//动作
    int num;//移进到哪个状态，或者用哪个产生式规约
};
```

全局变量：

```
vector<PF> PF_vector;//产生式
map<char, set<char> > first;//first集
map<char, set<char> > follow;//follow集
set<ITEM> DFA;//DFA项目集规范簇
vector<map<char, action>> predict_table_action;//SLR1分析表action
vector<map<char, int>> predict_table_goto;//SLR1分析表goto
```

```
vector<char> A_1;//符号栈
vector<int> A_2;//状态栈
vector<char> B;//剩余串
set<char> terminal;//所有的终结符+'$'
set<char> non_terminal;//所有的非终结符
int B_point = 0, input_len = 0;//B_point为输入串指针，input_len为输入串长度
```

函数：

```
ITEM::ITEM()//默认构造函数
ITEM::ITEM(int NO)//构造函数
bool ITEM::operator<(const ITEM& item)const//重载<操作符，以便使用 set
action::action(RS rs, int num) //构造函数
action::action()//默认构造函数
string action::ret_action()//返回该动作
void first_construction()//构造并输出 first 集
void add_follow(const char& ch1, const char& ch2)//将 ch1 的 follow 集加入到 ch2 的 follow 集中
void follow_construction()//构造并输出 follow 集
void DFA_construction()//构造 DFA
void predict_table_construction()//构造并输出 SLR1 预测分析表
void print_A()//输出分析栈
void print_B()//输出剩余串
void analyse()//预测分析控制程序
int main()
```

PS：关于每个函数内部详细实现源.cpp 中写好了详细的注释，我几乎对每一步操作的目的和函数的目的都写了详细的注释。

4、详细代码：

详见附件源.cpp，在此处不再赘述。

5、测试样例：

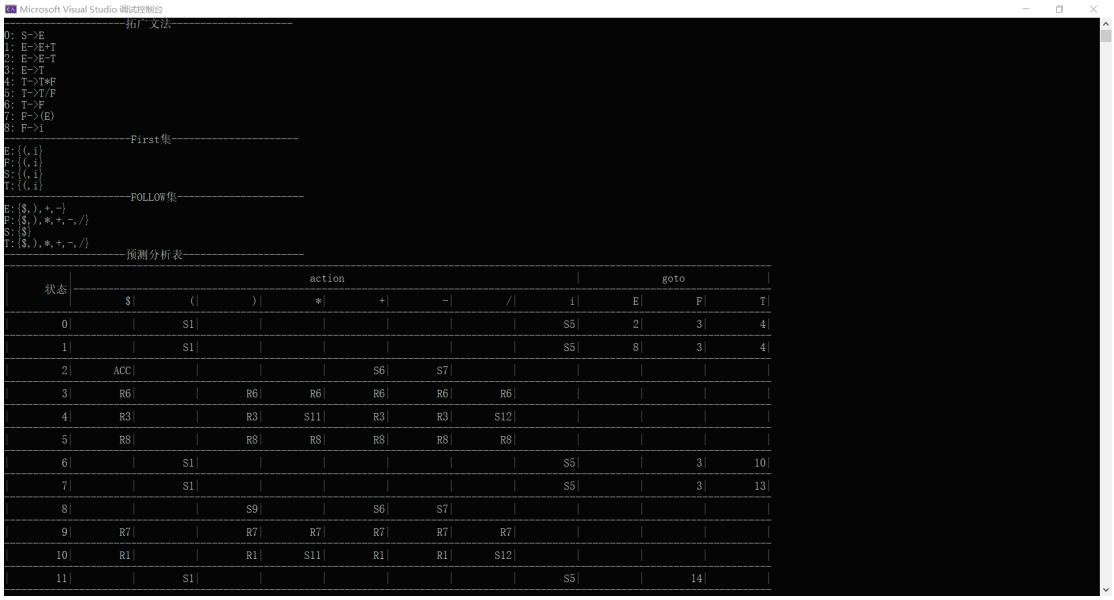
1、 输入：

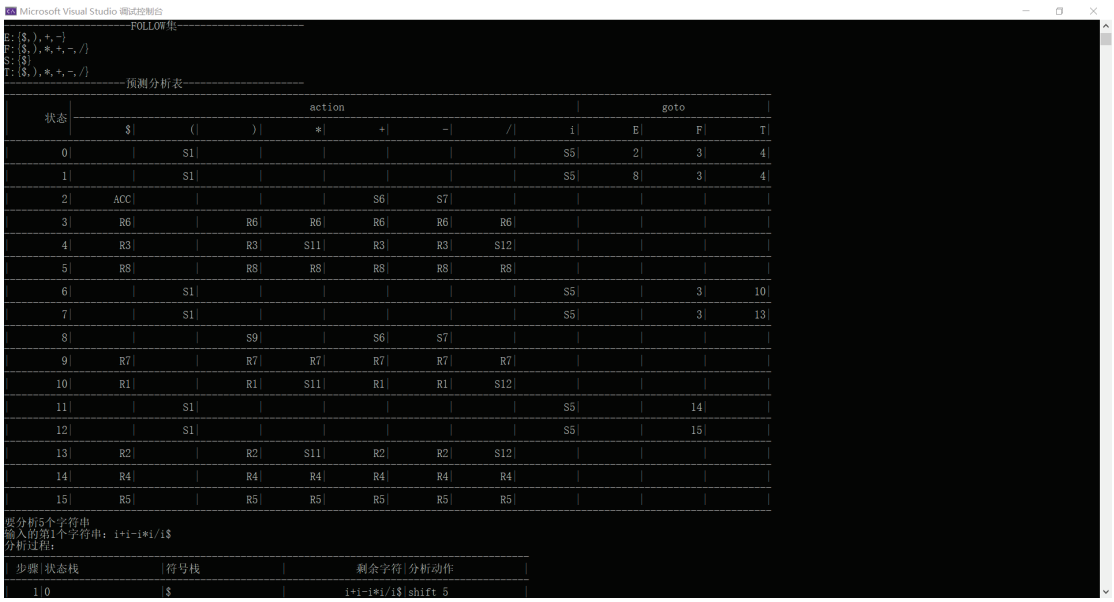
① 文件输入：in.txt

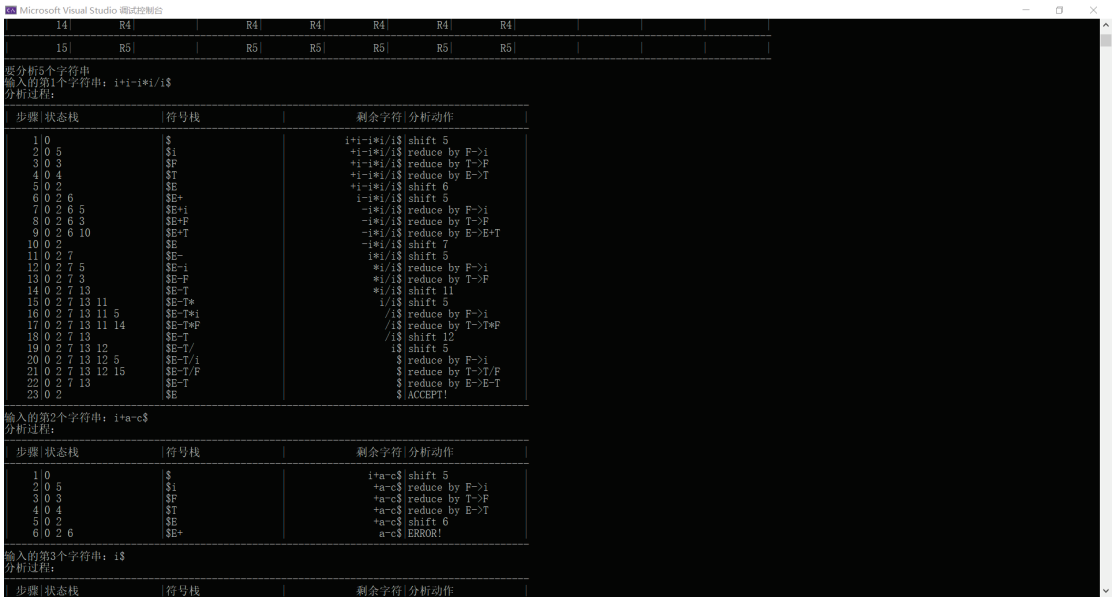
```
8
E->E+T
E->E-T
E->T
T->T*F
T->T/F
T->F
F->(E)
F->i
5
i+i-i*i/i$
```

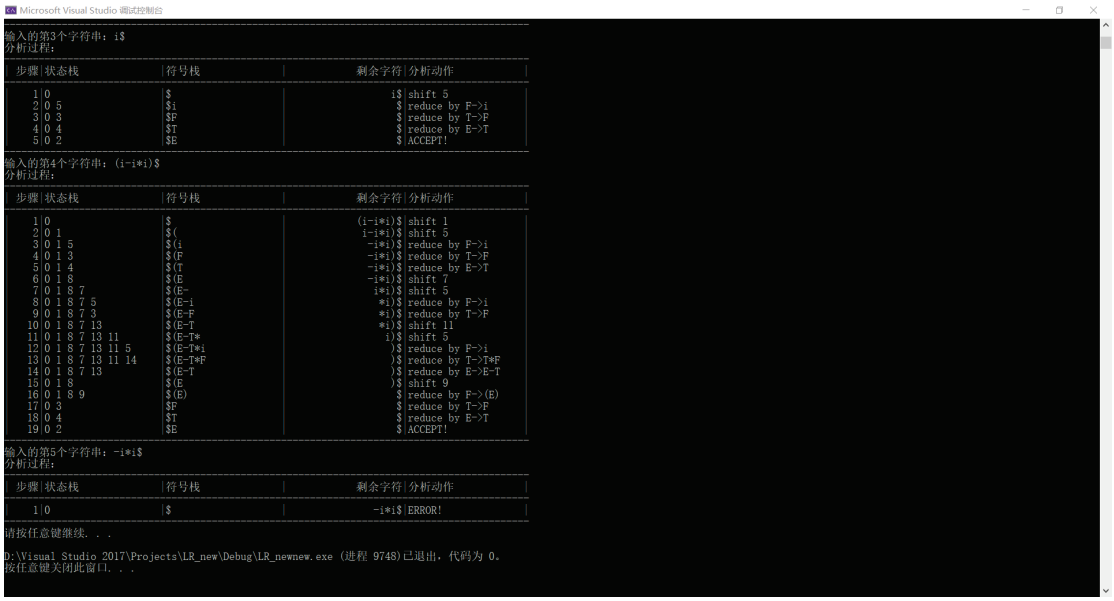
i+a-c$
i$
(i-i*i)$
-i*i$

2、输出：









6、实验总结

LR 语法分析器实验让我更深入的了解了求解含有递归的文法的 first 集和 follow 集，以及构造识别文法所有活前缀的 DFA 的算法和 LR 预测分析表的构造和预测分析程序的各种算法，平时不亲手写代码时感觉自己对算法很了解，但亲手写代码时还是感觉到了难度，只有亲身实践才能让自己对算法的理解更上一层楼。也顺便提高了我的编程能力，总之这次实验让我受益匪浅。