实验报告：图书管理系统

1. 环境概述

操作系统： Ubuntu 18.04

DBMS： MySQL

使用语言： Golang + SQL

使用软件： GoLand

2. 运行方式

2.1 运行准备

首先打开代码，修改

```
const (
    User     = "xxxx" //root of mysql
    Password = "xxxxxxxx" //password of root
    DBName   = "ass3"
)
```

*如果当前没有名为ass3的数据库，首先需要手动创建一个名为ass3的数据库，用来存放图书管理系统所需要的各种表格。

*注意，运行的时候会drop掉原来可能会产生冲突的表格（方便多次测试，实际中不需要），运行之前务必留意

接着，运行如下SQL代码，用来创建几个用户，用于模拟已经存在于系统中的用户。为了避免可能会产生的一系列冲突问题，这些代码并没有被放到图书管理系统的代码内部。（且在真实环境下也无需放到代码内部）

```
CREATE user '18300750006'@'localhost' identified by 'qwertyuiop';
GRANT Insert,Select,Update,Delete ON `ass3`.* TO `18300750006`@`localhost`;
CREATE user '17307130028'@'localhost' identified by 'qwertyuiop';
GRANT Insert,Select,Update,Delete ON `ass3`.* TO `17307130028`@`localhost`;
CREATE user '16301820009'@'localhost' identified by 'qwertyuiop';
GRANT Insert,Select,Update,Delete ON `ass3`.* TO `16301820009`@`localhost`;
```

修改并运行完以上内容之后，基本的运行所需条件就已经满足了。

***如果助教不能成功运行我的代码的话，烦请尽快联系我**

2.2 运行操作

在与代码同一目录下，运行

```
go test
```

即可运行测试样例。

而运行

```
go run library.go
```

即可在command line中与图书管理系统进行交互。

3 功能及实现

3.1 注意事项

1. 本实验及代码力求尽可能真实地还原一个真正的数据管理系统，但考虑到为了测试的方便直观以及助教的查验方便，在一些函数中可能会加入一些方便测试的代码及操作，在此提前进行说明，但在真实情况下无需加入这些代码及操作。加入部分代码及操作有：

- 对于createtables函数，在command line中与图书管理系统交互的时候，每一次交互都会重新创建一次表格，即本次做的操作及修改在退出系统后不会被保存，下次登陆进来后依旧是已经被设定好的表格，从而能够更加方便地测试这个系统，在实际操作中，并不需要每次都进行创建表格的操作，此处只是为了更加方便地让人感受这个系统的操作及验证这个系统的逻辑正确与否。

  如果助教想要自行对表格加入数据来测试函数的话，可以考虑

  - 使用command line交互，登录root，（例如）添加书后再使用List查看有没有添加成功（注意：退出交互后不保存本次的添加，包括添加账户（但账户实际上被创建了，只不过student表中该用户的信息会随着退出而丢失，进而导致错误），如果想要测试建议一次测完……或者如果表格已经成功创建的话，助教可以删除代码中的604-606行，这样就不会每次进入重新创建一次表格，不过仅限于command line交互，test中每次都会重新创建表格。当然助教也可以直接修改createtables函数逻辑，不建议这样去做。
  - 在test中对应函数部分写一些测试样例或者调用函数，并观察数据库的变化

- 同样createtables函数在创建表格的时候首先会删除数据库中可能已经存在的同名的表格，接着会在表格中插入一些初始数据，即每次运行开始的时候表格里的内容都是一样的，方便进行后续的测试，在真实环境下则根据实际情况来插入数据，在此列出四个表格初始化的内容

  （*四个表格具体的设计及属性在后续会介绍）

sname	sid	sex	age	major	st
lisa	16301820009	Female	18	EE	Forbidden
zhangsan	17307130028	Male	21	CS	Able
hanxy	18300750006	Male	20	CS	Able

bid	title	author	ISBN	state
73498	Crime and punishment	Fyodor Dostoyevsky	978-1-8571-5035-3	Borrowed
91067	Father Goriot	Honoré de Balzac	978-1-4264-0594-5	Available
20812	Madame Bovary	Gustave Flauberte	978-1-6740-7054-4	Available
60842	Miser	Molière	978-1-1206-4684-2	Borrowed
60843	Miser	Molière	978-1-1206-4684-2	Available
71425	Oliver Twist	Charles Dickens	978-1-8532-6012-4	Available
94681	One Hundred Years of …	Marquez	978-0-1411-8499-9	Borrowed
70826	Othello	W. William Shakespeare	978-0-5215-3517-5	Borrowed
71364	Pickwick Papers	Charles Dickens	978-0-1958-6308-6	Available
09728	The Hunger Games	Suzanne Collins	978-0-4390-2348-1	Borrowed

sid	bid	begintime	endtime	extendnum
18300750006	70826	2020-01-29	2020-05-29	3
18300750006	60842	2020-02-29	2020-04-29	1
17307130028	94681	2019-12-29	2020-03-29	2
17307130028	73498	2020-02-21	2020-04-21	1
17307130028	20812	2020-01-26	2020-03-26	1

sid	bid	begintime	endtime
18300750006	71364	2019-03-29	2019-05-29
18300750006	91067	2019-07-29	2019-08-29

- 与上面思路相同，在AddAccount函数中，我会首先删除可能存在的用户17301257636，因为后续我会创建一个id为17301257636的用户来进行成功创建账户的测试，在实际操作中不需要进行该步骤。在AddBook函数中则不需要做类似操作因为表格每次都会被重新初始化。
- 主要为以上几个操作，若还有类似操作则理由同上

2. 本管理系统有一些函数会跟当前系统时间有一定关系，比如自行新增的getarrears函数（以0.1元/本/天来计算逾期费用），不同的日期测出来的答案自然不同，同样在判断书是否逾期时候也会涉及到当前时间，在此进行说明，防止助教产生误解（不过已经尽可能避免因此原因导致的测试出现bug）。

*暂时就想到这么多的注意事项，如果助教在检查过程中遇到了一些不理解且我没有进行说明的地方，还烦请助教联系本人进行说明，避免产生误解。

3.2 表格设计

本人在数据库中设计了四个表格，下面是大致介绍

1. 学生（student）表格，用来存储学生姓名，学号（主键），性别等个人信息，其中学号作为主键，同时也作为学生的登陆账号，在此处学号利用了复旦的学号设计格式。
2. 书（book）表格，用来存放书名，书号（主键），作者等书的信息。其中书号在实际中最好设计成有一定含义的号码，比如前几位是馆号，中间是书架号，后面是书的类别及对应序号等，在此处为了测试直观方便，就设置成5位的号码，且在插入初始数据的时候并不考虑该种设计方式，只是简单地随便填写几个号码（随机而无意义），从而使得测试更加方便，看起来更加清楚，在实际操作中书号的设计会更有逻辑。
3. 借阅（borrowed）表格，用于存储正在进行中的借阅信息，包含学号（外键），书号（外键），借阅的开始时间，结束时间以及已经延长的次数（最多三次）。
4. 记录（record）表格，用于存储以往的借阅信息（不包括正在进行中的借阅），与借阅表格类似，只不过没有延长次数这一属性。

个人认为，这四个表格是一个图书管理系统设计的必需品，学生及书籍信息不必多说，借阅信息也是必须的，当然，借阅和记录两个表格可以合在一起，只不过这样会存储多余的延长信息，以及会使得读取表格的时候操作更加繁琐，更不方便，所以最后将其分开放置。在现实中，可能会根据实际情况增加一些新的表格，但在本图书管理系统中，则仅设计这四个基本表格，若有更多的需求，则在未来会更具实际情况进行设计。

3.3 功能实现

*功能实现不包括连接数据库，创建表格以及command line交互的实现

*在command line中新增了列出所有表格的功能（仅限root），因为过于简单，就没有放在此进行介绍以及添加测试

3.3.1 基本函数及作用

func (e xxxxxxxx) Error() string

该函数主要用于定义一种新的错误类型，由于在每个功能中都可能存在各种各样的错误，所以需要额外定义不同的错误类型用于返回，从而能够验证测试结果是否准确。举个例子，在学生借书的时候，调用函数时需要判断学生的学号是否有效，该学生是否有权限借书，书号是否有效，该书是否被借走，若不额外定义错误类型，在测试的时候如果学生没有成功借走书的话，返回的错误不能表现出没有成功借走书的原因，也就无法故意测试一些借书失败的样例来验证程序是否正确。也正因此，我利用该类型函数额外定义了多种错误类型，从而能够更好地进行测试。

func mustExecute(db *sqlx.DB, SQLs []string)

该函数参考自助教上一次布置的lab，用于让程序在数据库内执行某些SQL语句，利用该函数可以帮我们完成一系列对数据库的操纵，同时也可以简化代码，使得代码看起来更加简洁清爽。

func TestAddAccount(t *testing.T)

该函数为测试函数，用于测试对应的函数是否正确。

3.3.2 添加书

主函数：

func (lib *Library) AddBook(title, author, id, ISBN string) error

测试函数：

func TestAddbook(t *testing.T)

基本思想：

给定一系列的传入参数，即书籍的相关信息，运用mustExecute函数输出Insert语句，将书籍的信息插入进book表即可。

注意事项：

在添加书的时候，由于书籍的号码是主键，因此添加的书籍的号码不能与已有的书籍的号码重复，所以事先使用lib.db.QueryRow来查找book表中是否已经存在一个号码与打算给新书的号码相同的书，如果找到了，那么会输出“id已被占用”信息，并返回错误，若没找到则返回nil。

测试用例：

两本书，其中一本书赋予表中已有的id，另一本书则赋予一个没有被占有的id。若前者返回错误，后者返回nil则通过该函数的测试，否则报错。

3.3.3 删除书

主函数：

func (lib *Library) RemoveBook(id,reason string) error

测试函数：

func TestRemoveBook(t *testing.T)

基本思想：

本来打算直接将book表内对应信息的书删除，但是思考了一下发现这样的话会导致record里面记录的借阅信息错乱的问题，所以就决定将book表中对应书的信息改为”Removed”，这样就可以不对record表进行修改。对于borrowed表来说，既然书已经丢失，那必然是有学生告知图书馆或者图书馆自己的行为，即图书馆肯定知道书丢失，所以可以把有关这本书的正在借阅中的信息给删除，避免产生一系列的错误（比如学生归还该书然后该书又变成可用了）。综上所述，可以直接使用mustExecute函数输出delete语句删除borrowed表中有关该书的借阅信息并利用update语句将book表中该书的状态设置成“Removed“。对于record表，则没有修改的必要。

注意事项：

同样需要考虑删除的时候书库里是否实际存在这本书，如果书库中没有这本书，返回错误，否则返回nil。

测试用例：

两本书及其对应的删除理由，第一本是有效的书（即book表中存在这本书），第二本书是无效的书。如果前者的函数返回为nil，后者的函数返回为错误，则通过该测试样例。

3.3.4 添加账户

主函数：

func (lib *Library) AddAccount(sname,sid,sex,age,major,password string) error

测试函数：

func TestAddAccount(t *testing.T)

基本思想：

将学生的个人信息及账户密码作为参数传递给该函数，利用mustExecute函数和create语句直接创建一个可以访问数据库的账户，其中账号为学生的学号，密码则是学生自行输入的字符串，再赋予该账户一定的权限（SELECT,INSERT,DELETE,UPDATE），并将该账户对应的学生信息添加到student表中，状态设置成”Able“，表明可以进行借书操作。

注意事项：

所有账户的账号均为学生的学号，在创建新账户前，需要先检测是否已经存在一个该学号对应的账户，如果已经存在这个账户，则返回错误，如果没有存在这个账户，即可以新建账户，就返回nil表明可以成功创建账户。创建完账户后一定要给予权限，否则会导致学生登陆进自己的账户后无法进行类似借书，还书等操作。

测试用例：

与之前相同，一个准备新建的账户的账号是已有的某个账户的账号，另一个则是一个全新的账户的账号，对返回值进行判断即可测试该函数是否正确。

3.3.5 查询书

主函数：

func (lib *Library) QueryBook(manner, value string) error

测试函数：

func TestQueryBook(t *testing.T)

基本思想：

传入两个变量给该函数，其中一个变量是查询方式，例如ISBN，title等，另一个变量是查询值，比如ISBN的具体值等，利用lib.db.Query函数和select语句选出book表中符合要求的书，再利用循环遍历输出每一条书目的信息即可。

注意事项：

此处是查询对应信息的书籍，当没有查询结果的时候跟其他函数不同，一样可以完成查询，只不过查询的内容为空，当查询信息错误，比如输入的查询方式不是title，ISBN，author三者之一的时候，此函数会返回错误。

测试用例：

四个测试用例，两个正常但是方式不同的查询，一个查询方式错误的查询，另一个则是正常但是返回为空的查询，方式错误的查询返回错误，其余查询返回nil则表明通过该测试。

3.3.6 借书

*借书操作的函数本身并不考虑是否是学生账户，但是借书操作这个过程（即在样例测试中以及实际的command line交互中，均会连接该学生对应的数据库账户，学生执行借书操作的时候，会固定把自己的学号作为参数传进借书函数，实现学生登陆进自己的账户并进行借书），后续有关使用学生账户进行操作的相关函数与此处类同。

主函数：

func (lib *Library) BorrowBook(user,bookid string) error

测试函数：

func TestBorrowBook(t *testing.T)

基本思想：

给该函数传入学号和书号两条信息，利用mustExecute函数执行insert语句，向borrowed表中插入一条借阅信息，同时将book表中对应的书的状态改为”Borrowed“即可。

注意事项：

需要对传入的学号，书号做存在性检测，如果不存在则需要报出相应的错误，同时，即使都存在，但如果书当前的状态不是”Available“，也将返回错误，拒绝学生的借书请求。

测试用例：

三个测试样例，一个是合法的操作，一个是信息错误的操作，还有一个是书的状态不是”Available“的操作，第一个操作返回nil，后面两个操作返回对应的错误信息即表明通过该测试。

3.3.7 查询借阅历史

主函数：

func (lib *Library) BorrowHistory(user string) error

测试函数：

func TestBorrowHistory(t *testing.T)

基本思想：

输入学生的学号，利用lib.db.Query和select语句查询对应学生在record表内的借书信息，循环遍历输出每条信息即可。

注意事项：

需要判断该学号是否合法，即是否存在学号为输入的学号的学生，若不存在则返回错误。利用lib.db.Query操作先查询该学生的信息即可实现这一目的。

当该学号对应的学生没有借阅信息的时候，依旧正常返回，只不过借阅信息为空而已。

测试用例：

三个测试用例，一个是有借阅信息的学生的学号，一个是无借阅信息的学生的学号，一个是错误的学号，前两者输出对应的信息并返回nil，最后一个返回错误表明通过该测试。

3.3.8 查询借阅中的书

主函数：

func (lib *Library) BorrowNow(user string) error

测试函数：

func TestBorrowNow(t *testing.T)

基本思想：

与3.3.7相同，只不过查询的表换成borrowed表

注意事项：

与3.3.7相同

测试用例：

与3.3.7相同

3.3.9 查询归还日期

主函数：

func (lib *Library) CheckDeadline(user,bid string) error

测试函数：

func TestCheckDeadline(t *testing.T)

基本思想：

输入学号和书号，利用lib.db.Query和select语句选出对应的书的归还日期并输出即可。

注意事项：

需要对学号和书号进行存在性判断，同时也需要验证该学号对应的学生是否借了该书号对应的书，如果借了的话就正常输出，返回nil，如果没借的话就返回错误，并输出相关的提示信息。

测试用例：

两组测试样例，一组测试样例是可以查询出结果的合法样例，另一组测试样例则是学号与书号无法匹配上（即该同学并没有借这本书）的测试样例，前者返回nil，后者返回错误则通过该测试。

3.3.10 延长归还日期

主函数：

func (lib *Library) ExtendDeadline(user,bid string) error

测试函数：

func TestExtendDeadline(t *testing.T)

基本思想：

将学号和书号作为参数传入，利用lib.db.Query和select语句选出该书的归还日期以及延长次数，接着对延长次数进行判断，如果次数>=3次，就返回错误，表明无法延长，如果次数<3次，则利用mustExecute函数和update语句将borrowed表中的归还时间+一个月，延长次数+1即可。

注意事项：

与3.3.9相同，需要判断学号和书号的存在性及学号对应的学生是否借了该书号对应的书，同时需要注意这与无法延长虽然都返回错误，但错误类型不同。

测试用例：

三组测试样例，一组是可以正常延长的合法样例，一组是延长次数>=3次后无法继续延长的样例，一组是书号与学号不匹配的测试样例，第一个返回nil，第二个返回延长错误（ErrExtend），第三个返回匹配错误（ErrDeadline）即通过该测试。

3.3.11 查询逾期书籍

主函数：

func (lib *Library) CheckOverdue(user string) error

测试函数：

func TestCheckOverdue(t *testing.T)

基本思想：

在3.3.9的筛选基础上去除书号判断相等，多增加一个判断条件，即归还时间<当前时间即可。

注意事项：

与3.3.9类同

测试用例：

三组测试样例，一组有逾期书籍的学生，一组没有逾期书籍的学生，还有一组为不存在的学号。前两者返回nil，第三者返回错误即通过测试样例。

3.3.12 还书

主函数：

func (lib *Library) ReturnBook(user,bookid string) error

测试函数：

func TestReturnBook(t *testing.T)

基本思想：

利用mustExecute函数，将borrowed表中的借阅信息删除，再将该信息插入进record表中，最后把book

表中对应的书的状态设置成”Available“即可。

注意事项：

需要判断学号和书号的存在性以及该学号的同学是否借了对应书号的书，并返回对应的错误信息。

测试用例：

两组测试用例，一个是正常的归还测试，还有一个是学号和书号不匹配的测试（即该同学并没有借所给的书），当前者返回nil，后者返回错误的时候，表明测试成功，函数正确。

3.3.13 冻结账户

主函数：

func (lib *Library) SuspendAccount(user string) error

测试函数：

func TestSuspendAccount(t *testing.T)

基本思想：

在3.3.9和3.3.10的基础上，对过期书籍的数目进行计数，当发现过期的书的总数目>=3时，利用mustExecute函数和update语句将student表中的学生的状态信息改为”Forbidden“，即可以禁止学生进行借书等操作。

注意事项：

需要判断所给的学生的学号是否合法以及该学生是否有超过3本书过期，如果学号不合法或者学生过期的书不足3本，则返回对应的错误信息，否则返回nil。

测试用例：

三组测试样例，第一组测试样例所给的学生没有3本及3本以上的书过期，第二组测试样例是非法的学号，第三组测试样例可以正常地冻结账户。第一、二组返回对应的错误信息，第三组返回nil则表明通过测试。

3.3.14 解冻账户

*此功能为新加入功能，用于解冻学生账户，解冻条件为：1、该账户处于冻结状态 2、该账户已经没有过期的书 满足这两个条件即可解冻该账户。

主函数：

func (lib *Library) FreeAccount(user string) error

测试函数：

func TestFreeAccount(t *testing.T)

基本思想：

与3.3.13一致，更改student表中对应学生的状态即可。

注意事项：

与3.3.13基本一致，不过此处没有单独对解冻未被冻结的账户这一情况做讨论。

测试用例：

三组测试样例，第一组为学号非法的样例，第二组为成功解冻的样例，第三组为过期书没有还完而不能解冻的样例。一、三返回对应错误信息，二返回nil即通过测试。

3.3.15 查询逾期费用

*此功能为新加入功能，用于计算学生需要支付逾期费用的数目（0.1元/本/天）

主函数：

func (lib *Library) Getarrears(user string) error

测试函数：

func TestGetarrears(t *testing.T)

基本思想：

对于查询到的逾期书籍（与3.3.9和3.3.10方式基本相同），计算出逾期天数，并用0.1元/本/天的单价计算累加得到总共的逾期费用并输出。

注意事项：

判断学号是否合法即可。

测试用例：

三组测试用例，第一组为非法的学号，第二组为有逾期的书的学生，第三组为没有逾期的书的学生，第一组返回错误而第二、三组返回nil，且第二、三组成功输出了对应的费用则表明通过测试。

4. 命令行交互

4.1 界面展示

根据之前给的方式开启命令行交互之后，出现以下界面

```
Welcome to the Library Management System!
Please input your account, if you don't have one, please input 'create' to creat
e a new account.
```

在这个界面，如果你想要登录管理员账户，输入root，如果你想要登录学生账户，输入账户号，如果没有账户，输入create开始执行addaccount等相关操作，这里就不具体展示了。

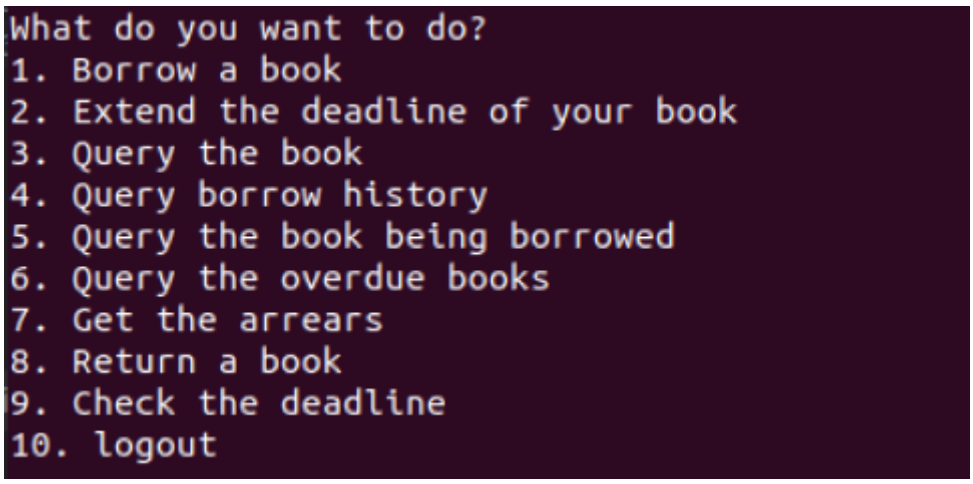
*补充一点，如果创建账户后退出了系统，则下一次登录进去后因为表格重置的原因，会导致能连进数据库，但是student表中没有该用户的信息，进而无法完成部分操作。如果需要避免此问题，第一次创建表格后把交互部分的代码中创建表格的代码删除即可，这只是方便测试，实际中自然不会这么写。（604-606行）

以下是登陆学生账户后出现的界面，需要注意的是，root用户的登陆界面与学生登录后的界面是不一样的，各有各的操作，具体助教可以去试一下。

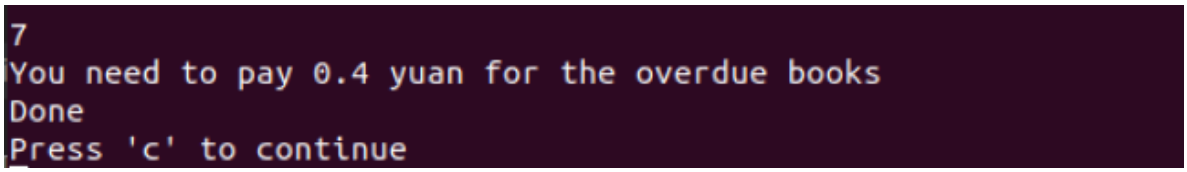
```
18300750006
Please input your password.
qwertyuiop
Successfully connect to the system
```

需要补充一点，在此处无论成功与否都会跳转到接下来执行操作的选择界面，但是如果密码输入错误的话，即使进入那个界面，因为无法成功连接数据库，也是无法执行接下来的操作的。主要是自己实在是不想再额外添加一个判断密码是否正确的功能……

*以下界面是学生能看到的界面，root账户界面与其不同



输入对应的数字即会出现对应的操作，输入10即可退出整个系统，以输入7为例：



会返回学生需要支付的逾期费用，按c键即可返回到之前"What do you want to do?"界面，这便是命令行交互的大致演示，具体助教可以自行尝试。

*该部分尽可能避免了一部分bug，如果本人考虑不周产生了一些意料之外的bug，烦请助教理解。

4.2 具体实现

该部分的具体实现没有特别多需要提及的地方，因为比较简单，只需根据一系列if，switch语句将输入进行解析，转化为对应的操作，执行完后再输出相应结果即可，在此处因为报告篇幅原因就不展开了。

5. 更多思考

一些感想

因为时间原因（毕竟还要准备不知道什么时候的期中考试，同期还有一个lab和一个pj），这个系统做得比较简陋，但整体来说，已经能较为完整地实现一个图书管理系统，自己也是比较满意的。通过这个lab，自己的写代码水平也有所提高，对于数据库也有了新的认识，也大概懂得如何去设计一个管理系统，收获还是很多的。

一些改进

该管理系统还是有许多可以进行完善和提升的地方，比如上文中提到的很多因为时间原因没有完善的功能，比如可以再额外添加一些其他功能，或者可以做一个前端等等等等，希望以后自己有机会改进这个图书管理系统。

***如果本人哪里写的有问题，或者操作逻辑上有比较严重的bug，烦请助教指出，谢谢**