北京工业大学

2020 - 2021 学年 第1学期

信息学部 计算机学院

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称： | 数据结构课程设计 | | |
| 报告性质： | 实验报告 | | |
| 学号： | 18074102 | 姓名： | 左帅 |
| 任课教师： | 王众 | 课程性质： | 学科基础必修课 |
| 学分： | 2 | 学时： | 60 |
| 班级： | 180701 | 成绩： |  |
| 教师评语： |  | | |

|  |  |  |
| --- | --- | --- |
| 需求分析 | 根据题中需求，提供功能划分说明 |  |
| 设计 | 逻辑结构、存储结构设计、算法描述 |  |
| 使用说明 | 界面是否友好 |  |
| 总结 | 是否感悟有收获 |  |
| 摘要 | 考察文字抽象能力 |  |
| 格式 | 是否有目录、页号 |  |

2020年 12 月 17 日

目录

[1需求分析 3](#_Toc59177510)

[1.1功能概述 3](#_Toc59177511)

[1.1.1基础功能 3](#_Toc59177512)

[1.1.2拓展功能 3](#_Toc59177513)

[1.2需要处理的数据 3](#_Toc59177514)

[1.3程序开发运行环境 3](#_Toc59177515)

[1.4用户界面设计 4](#_Toc59177516)

[2数据结构设计 5](#_Toc59177517)

[2.1主要数据结构定义 5](#_Toc59177518)

[2.1.1逻辑结构 5](#_Toc59177519)

[2.1.2存储结构 5](#_Toc59177520)

[2.2整体结构及模块功能描述 5](#_Toc59177521)

[2.3各模块功能描述 5](#_Toc59177522)

[2.3.2 AVL类（AVLTree） 5](#_Toc59177523)

[2.3.3 链表类（linked\_list） 5](#_Toc59177524)

[2.3.4 有序链表类（sorted\_linked\_list） 5](#_Toc59177525)

[2.3.5 GUI类（Ui\_FSS, Ui\_review, MyWindow, Review, MyQLabel） 5](#_Toc59177526)

[2.3.6 搜索类（search） 5](#_Toc59177527)

[2.4核心算法流程图 6](#_Toc59177528)

[3详细设计 7](#_Toc59177529)

[3.1各模块设计 7](#_Toc59177530)

[3.1.1 film 7](#_Toc59177531)

[3.1.2 GUI 7](#_Toc59177532)

[3.1.3 AVLTree 8](#_Toc59177533)

[3.1.4 链表类 8](#_Toc59177534)

[3.2各模块核心函数 9](#_Toc59177535)

[3.2.1 avl树创建函数 9](#_Toc59177536)

[3.2.2 有序链表的插入 10](#_Toc59177537)

[3.2.3 搜索内容 11](#_Toc59177538)

[3.3详细代码 12](#_Toc59177539)

[3.3.1 AVLTree.py 12](#_Toc59177540)

[3.3.2 Film.py 19](#_Toc59177541)

[3.3.3 GUI.py 19](#_Toc59177542)

[3.3.4 linkedlist.Py 39](#_Toc59177543)

[3.3.5 sortredlinkedlist.py 41](#_Toc59177544)

[3.3.6 Search.py 44](#_Toc59177545)

[4用户手册及测试结果 49](#_Toc59177546)

[1、排序演示 49](#_Toc59177547)

[2、添加评论 50](#_Toc59177548)

[3、交叉查询演示 51](#_Toc59177549)

[4、访问网页 52](#_Toc59177550)

[5、修改文件 53](#_Toc59177551)

[5总结提高 54](#_Toc59177552)

[5.1体会与收获 54](#_Toc59177553)

[5.2问题与解决 54](#_Toc59177554)

[5.3评价 54](#_Toc59177555)

[6参考文献 54](#_Toc59177556)

# 1需求分析

## 1.1功能概述

### 1.1.1基础功能

本程序需要实现一个电影检索系统，拥有以下功能：

1、设计实现一个图形界面，能根据不同的分类如豆瓣评分、名著改编、地域、剧情、导演、演员、语言查询到想要的电影，并且各种分类之间要有交叉。

2、设计实现用户可以通过输入电影名，直接查询电影

3、可以按照最新、最热排行查询。

### 1.1.2拓展功能

1、可以让用户添加评语

2、可以点击访问外部链接

## 1.2需要处理的数据

针对三个基础功能，对于功能一需要有电影的评分、地域、剧情、导演、演员、语言这几种数据，其中地域,剧情,语言和演员包含多个数据。电影文件存储于filmlist.txt文件中。数据总计250部电影。

## 1.3程序开发运行环境

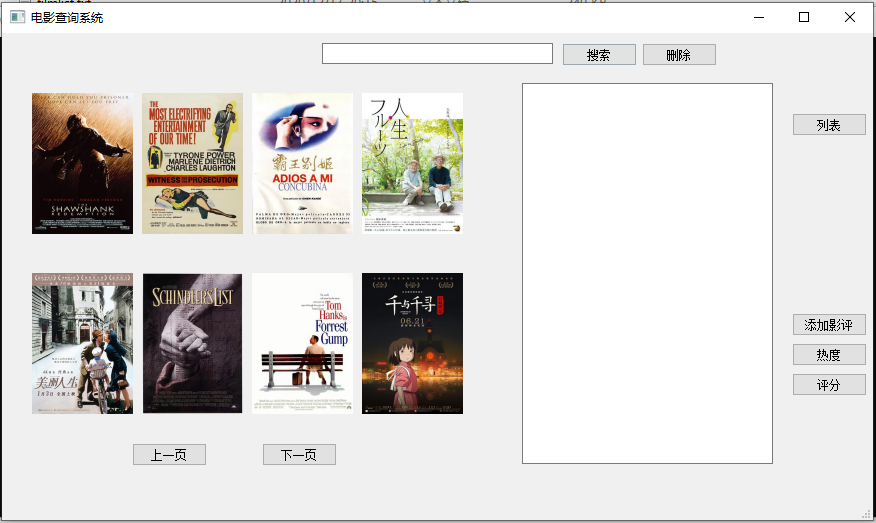
开发环境：

* Visual Studio Code 1.52.1
* Python 3.8.5 64bit

运行环境：

* Windows 10 Professional 64bit 18363

## 1.4用户界面设计



在基础功能1的查找中，用户可以输入电影名，导演名，类别等对电影进行筛选。用户同时也可以点击左边的区域查看电影的信息。所有搜索得到的电影和点击查看的电影热度均会增加。右方的文本框用于信息显示，热度与评分用于对应的排序，添加影评可以与许用户添加影评。

# 2数据结构设计

## 2.1主要数据结构定义

### 2.1.1逻辑结构

对于名称查找使用的是树形结构，同时对于名称和热度排序使用的是线性结构。

### 2.1.2存储结构

电影存储于链表与树中，因此存储结构为链式存储。

## 2.2整体结构及模块功能描述

本电影查询系统又电影节点类、链表类、AVL树类、GUI类组成，节点类主要用于存储数据和对数据操作，链表类、AVL树类、GUI类主要用于实现各种功能。

## 2.3各模块功能描述

2.3.1电影节点类（film）

链表中的节点，负责储存电影的信息。

数据：电影名（name），导演（director），演员（actors），类型（types），豆瓣评分（rating），地域（region），语言（language），上架时间（time）,剧情（plot）, 热度（click\_times）,影评（review），链接（url）。

电影节点类中包含方法clicked()用于使热度自增。

### 2.3.2 AVL类（AVLTree）

用于创建包含节点内容是电影的AVL树，并对相应的内容进行查找

### 2.3.3 链表类（linked\_list）

用于创建包含节点内容是电影的链表，并对相应的内容进行查找

### 2.3.4 有序链表类（sorted\_linked\_list）

用于创建包含节点内按权值weight有序排布的链表，从而方便进行时间排序与热度排序。

### 2.3.5 GUI类（Ui\_FSS, Ui\_review, MyWindow, Review, MyQLabel）

用于GUI的生成。

### 2.3.6 搜索类（search）

用于处理输入的搜索内容，并返回对应的搜索得到的内容

## 2.4核心算法流程图

开始

读取文件内容

将文件信息储存到AVL树与链表中

是否点击了电影图片

将电影信息，影评，链接显示在信息框

是

否

是否点击了搜索按钮

判断其为电影名或者为其他如导演并进行搜索

是

否

是否点击了热度或评分

根据对应排序方法切换排序好的链表

是

是否点击影评按钮

是

输入搜索的内容

输入影评

为对应的电影添加影评

是否退出

结束

是

否

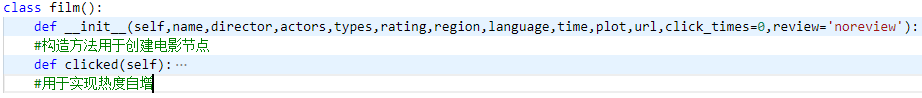
否

+

# 3详细设计

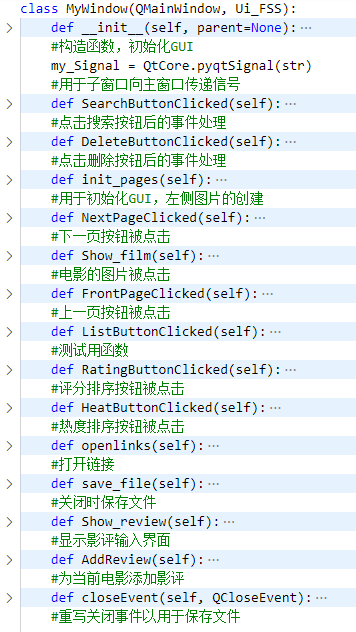
## 3.1各模块设计

### 3.1.1 film



### 3.1.2 GUI





### 3.1.3 AVLTree

class AVLTree:

    class film\_node:

    #节点类

    def \_\_init\_\_(self):

    #构造函数，声明树的根节点self.root，设置树的节点数为0

    def get\_size(self):

#获取树的节点数

    def add\_from\_root(self,film)：

    def add(self, node,film):

    #添加节点

    def right\_rotate(self, y):

    #右旋

    def left\_rotate(self, y):

    #左旋

    def get\_node(self, node, film\_name)：

#搜索节点，根据node的film\_name值

    def preorder\_traversal\_save(self):

#先序周游并将结果储存在文件中

    def get\_film\_from\_node\_llist(self,film\_name):

    #获取搜索节点的film类

    def get\_film\_from\_node\_print(self,film\_name):

    #获取搜索节点的film类，并打印其内容

    def contains(self, film\_name):

    #返回是否包含对应的电影名

    def preorder\_traversal(self,node)：

    def root\_preorder\_traversal(self):

    #先序周游

    def root\_preorder\_traversal\_search(self,types):

    def preorder\_traversal\_search(self,node,search\_str):

#先序周游，并比对电影的类型

    def remove(self, film\_name):

    def minimum(self, node):

    def \_remove(self, node, film\_name):

#节点的删除

### 3.1.4 链表类

class sorted\_linked\_list():

    def \_\_init\_\_(self):

    #构造函数

    def insert\_by\_rating(self,film):

    #根基电影的评分顺序插入节点

    def insert\_by\_clicks(self,film):

    #根基电影的热度顺序插入节点

    def \_show(self,node):

    def show(self):

    #返回已排序的内容

    def \_delete(self,film\_name,node):

    def delete\_null\_node(self):

#删除冗余节点

    def delete(self,film\_name):

    #节点的删除

class linked\_list():

    def \_\_init\_\_(self):

#构造函数

    def \_append(self,film,node):

    def append(self,film):

#尾插

    def is\_empty(self):

#是否为空

    def delete(self,film\_name):

    #节点的删除

    def \_show(self,node):

    def show(self):

#返回节点的内容

## 3.2各模块核心函数

### 3.2.1 avl树创建函数

def add(self, node,film):

        if not node:

            self.size += 1#若节点为空，则创建一个节点

            return self.film\_node(film)

        elif node.film\_name > film.name:

            node.left = self.add(node.left,film)

        else:

            node.right = self.add(node.right,film)

        # 更新height,获取平衡因子

        node.height = 1 + max(self.get\_height(node.left), self.get\_height(node.right))

        banlance\_factor = self.get\_balance\_factor(node)

        # 维护平衡性，共四种情况LL,RR,LR,RL

        #LL

        if banlance\_factor > 1 and self.get\_balance\_factor(node.left) >= 0:

            return self.right\_rotate(node)

        #RR

        if banlance\_factor < -1 and self.get\_balance\_factor(node.right) <= 0:

            return self.left\_rotate(node)

        #LR

        if banlance\_factor > 1 and self.get\_balance\_factor(node.left) < 0:

            node.left = self.left\_rotate(node.left)

            return self.right\_rotate(node)

        #RL

        if banlance\_factor < -1 and self.get\_balance\_factor(node.right) > 0:

            node.right = self.right\_rotate(node.right)

            return self.left\_rotate(node)

        return node

该函数作用是创建AVL树，在将节点加入树后，根据平衡因子的值从而进一步的调整平衡。

### 3.2.2 有序链表的插入

def insert\_by\_rating(self,film):

        self.symbol=1

        if self.head is None:

            self.head=self.node(float(film.rating))

            self.head.filmlist.append(film)

        else:

            p=self.head

            q=self.head

            while(float(p.weights)>float(film.rating)):

                q=p

                p=p.next

                if p is None:

                    break

            if q==p:

                if(p.weights!=film.rating):

                    self.head=self.node(film.rating)

                    self.head.filmlist.append(film)

                    self.head.next=p

                    return

            if p is not None:

                if float(p.weights)<float(film.rating):

                    q.next=self.node(float(film.rating))

                    q.next.filmlist.append(film)

                    q.next.next=p

                else:

                    p.filmlist.append(film)

            else:

                q.next=self.node(float(film.rating))

                q.next.filmlist.append(film)

                q.next.next=p

该函数作用是根据节点权值将其插入到有序链表中，并使链表仍然有序。

### 3.2.3 搜索内容

def search(self,search\_str):

        self.search\_word = search\_str.split()

        i = 0

        for word in self.search\_word:

            if self.all\_types.count(word) < 1 and self.directors.count(word) < 1 and self.actors.count(word) < 1:

                i = 1

        if i == 0:

            str1=self.avl.root\_preorder\_traversal\_search(search\_str)

            if len(self.avl.need\_refresh\_list)!=0:

                self.refresh\_list(self.clicks\_sort, self.avl.need\_refresh\_list)

                return str1

        elif search\_str == 'list':

            return self.list\_all()

        elif search\_str == 'types':

            return self.show\_types()

        elif search\_str == 'directors':

            return self.show\_directors()

        elif search\_str == 'actors':

            return self.show\_actors()

        elif search\_str == 'ratingsort':

            return self.rating\_sort.show()

        elif search\_str == 'heatsort':

            return self.clicks\_sort.show()

        elif re.match(r'delete .\*',search\_str):

            self.delete(search\_str.split()[1])

        else:

            #searched\_film = self.avl.get\_film\_from\_node(search\_str)

            str1=self.avl.get\_film\_from\_node\_llist(search\_str)

        if len(self.avl.need\_refresh\_list)!=0:

            self.refresh\_list(self.clicks\_sort, self.avl.need\_refresh\_list)

            return str1

        return ''

该函数作用是根据搜索的字符串返回相应的内容。

## 3.3详细代码

### 3.3.1 AVLTree.py

from Film import film

from linkedlist import linked\_list

class AVLTree:

    class film\_node:

        def \_\_init\_\_(self, film):

            self.film\_name = film.name

            self.left = None

            self.right = None

            self.height = 1

            self.film = film

            self.strs=''

    def \_\_init\_\_(self):

        self.root = None

        self.size = 0

        self.need\_refresh\_list=[]

    def get\_size(self):

        return self.size

    def add\_from\_root(self,film):

        self.root=self.add(self.root,film)

    def add(self, node,film):

        if not node:

            self.size += 1#若节点为空，则创建一个节点

            return self.film\_node(film)

        elif node.film\_name > film.name:

            node.left = self.add(node.left,film)

        else:

            node.right = self.add(node.right,film)

        # 更新height,获取平衡因子

        node.height = 1 + max(self.get\_height(node.left), self.get\_height(node.right))

        banlance\_factor = self.get\_balance\_factor(node)

        # 维护平衡性，共四种情况LL,RR,LR,RL

        #LL

        if banlance\_factor > 1 and self.get\_balance\_factor(node.left) >= 0:

            return self.right\_rotate(node)

        #RR

        if banlance\_factor < -1 and self.get\_balance\_factor(node.right) <= 0:

            return self.left\_rotate(node)

        #LR

        if banlance\_factor > 1 and self.get\_balance\_factor(node.left) < 0:

            node.left = self.left\_rotate(node.left)

            return self.right\_rotate(node)

        #RL

        if banlance\_factor < -1 and self.get\_balance\_factor(node.right) > 0:

            node.right = self.right\_rotate(node.right)

            return self.left\_rotate(node)

        return node

    def get\_height(self, node):

        if not node:

            return 0

        return node.height

    def get\_balance\_factor(self, node):

        if not node:

            return 0

        return self.get\_height(node.left) - self.get\_height(node.right)

    def right\_rotate(self, y):

        x = y.left

        T3 = x.right

        # 右旋转

        x.right = y

        y.left = T3

        # 更新height

        y.height = max(self.get\_height(y.left), self.get\_height(y.right)) + 1

        x.height = max(self.get\_height(x.left), self.get\_height(x.right)) + 1

        return x

    def left\_rotate(self, y):

        x = y.right

        T2 = x.left

        x.left = y

        y.right = T2

        # 更新height

        y.height = max(self.get\_height(y.left), self.get\_height(y.right)) + 1

        x.height = max(self.get\_height(x.left), self.get\_height(x.right)) + 1

        return x

    def get\_node(self, node, film\_name):

        if not node:

            return

        if node.film\_name == film\_name:

            return node

        elif node.film\_name > film\_name:

            return self.get\_node(node.left, film\_name)

        else:

            return self.get\_node(node.right, film\_name)

    def get\_film\_from\_node(self,film\_name):

        s = self.get\_node(self.root, film\_name)

        if s is not None:

            return s.film

        else:

            return film('noname','norating','noregion','nointro','notypes','nosuchdirector','noactor','nolanguage','notime','nourl','nope')

    def preorder\_traversal\_save(self):

        film=open("filmlist.txt", "w",encoding='utf-8')

        self.\_preorder\_traversal\_save(self.root,film)

        film.close()

    def \_preorder\_traversal\_save(self,node,film):

        if node is not None:

            #filmname,director,actors,types,ratings,regions,languages,date,intro

            film.write(node.film.name)

            film.write('#')

            film.write(node.film.director)

            film.write('#')

            film.write('/'.join(node.film.actors))

            film.write('#')

            film.write('/'.join(node.film.types))

            film.write('#')

            film.write(node.film.rating)

            film.write('#')

            film.write('/'.join(node.film.region))

            film.write('#')

            film.write('/'.join(node.film.language))

            film.write('#')

            film.write(node.film.time)

            film.write('#')

            film.write(node.film.plot)

            film.write('#')

            film.write(node.film.url)

            film.write('#')

            film.write(str(node.film.click\_times))

            film.write('#')

            film.write('/'.join(node.film.review))

            film.write('\n')

            self.\_preorder\_traversal\_save(node.left,film)

            self.\_preorder\_traversal\_save(node.right,film)

    def get\_film\_from\_node\_llist(self,film\_name):

        s = self.get\_node(self.root, film\_name)

        llist = linked\_list()

        self.need\_refresh\_list=[]

        if s is not None:

            llist.append(s.film)

            s.film.clicked()

            self.need\_refresh\_list=[s]

            return llist

        else:

            return film('noname','norating','noregion','nointro','notypes','nosuchdirector','noactor','nolanguage','notime', 'nourl','nope')

    def get\_film\_from\_node\_print(self,film\_name):

        self.searched\_film = self.get\_film\_from\_node(film\_name)

        if self.searched\_film.name != 'noname':

            print(self.searched\_film.name,self.searched\_film.director,self.searched\_film.types)

            self.searched\_film.clicked()

            self.need\_refresh\_list=[self.searched\_film]

            return '电影名称:'+self.searched\_film.name+ \

            '\n导演:'+self.searched\_film.director+ \

            '\n演员:'+' '.join(self.searched\_film.actors)+ \

            '\n类型:'+' '.join(self.searched\_film.types)+ \

            '\n上映日期: '+self.searched\_film.time+ \

            '\n语言:'+' '.join(self.searched\_film.language)+ \

            '\n区域:'+' '.join(self.searched\_film.region)+ \

            '\n评分:'+self.searched\_film.rating+ \

            '\n剧情梗概:'+self.searched\_film.plot+'\n'

        else:

            print('No Such Film!')

            return 'No Such Film!'

    def contains(self, film\_name):

        return self.get\_node(self.root, film\_name) is not None

    def preorder\_traversal(self,node):

        if node is not None:

            print(node.film\_name,node.film.director,node.film.types)

            self.strs=self.strs+node.film\_name+'  ['+' '.join(node.film.types)+']\n'

            self.preorder\_traversal(node.left)

            self.preorder\_traversal(node.right)

    def root\_preorder\_traversal(self):

        self.strs=''

        self.preorder\_traversal(self.root)

        return self.strs

    def root\_preorder\_traversal\_search(self,types):

        self.flag = 0

        self.strs = ''

        self.llist = linked\_list()

        self.need\_refresh\_list=[]

        self.preorder\_traversal\_search(self.root,types)

        if self.flag == 0:

            print("No Such Film!")

            return 'No Such Film!'

        #return self.strs

        return self.llist

    def preorder\_traversal\_search(self,node,search\_str):

        if node is not None:

            i = 1

            strs=search\_str.split()

            for string in strs:

                if node.film.types.count(string) < 1 and node.film.director!=string and node.film.actors.count(string) < 1:

                    i = 0

            if i == 1:

                self.flag = 1

                #print(node.film\_name,node.film.director,node.film.actors,node.film.types)

                node.film.clicked()

                self.need\_refresh\_list.append(node)

                self.llist.append(node.film)

                #self.strs =self.strs + '电影名称：'+node.film.name+ \

                #'\n导演:'+node.film.director+ \

                #'\n演员:'+' '.join(node.film.actors)+ \

                #'\n类型:'+' '.join(node.film.types)+ \

                #'\n上映日期: '+node.film.time+ \

                #'\n语言:'+' '.join(node.film.language)+ \

                #'\n区域:'+' '.join(node.film.region)+ \

                #'\n评分:'+node.film.rating+ \

                #'\n剧情梗概:'+node.film.plot+'\n\n'

            self.preorder\_traversal\_search(node.left,search\_str)

            self.preorder\_traversal\_search(node.right,search\_str)

    def remove(self, film\_name):

        node = self.get\_node(self.root, film\_name)

        if node:

            self.root = self.\_remove(self.root, film\_name)

        # 删除以node为根的BST中键值为key的节点，递归算法

        # 返回删除节点后的新的BSTMap的根

    def minimum(self, node):

        if not node.left:

            return node

        return self.minimum(node.left)

    def \_remove(self, node, film\_name):

        if not node:

            return

        # 递归条件

        if node.film\_name > film\_name:

            node.left = self.\_remove(node.left, film\_name)

            ret\_node = node

        elif node.film\_name < film\_name:

            node.right = self.\_remove(node.right, film\_name)

            ret\_node = node

        else:  # node.film\_name == film\_name

            if not node.left:

                right\_node = node.right

                node.right = None

                self.size -= 1

                ret\_node = right\_node

            elif not node.right:

                left\_node = node.left

                node.left = None

                self.size -= 1

                ret\_node = left\_node

            else:

                successor = self.minimum(node.right)

                successor.right = self.\_remove(node.right, successor.film\_name)

                successor.left = node.left

                node.left = node.right = None

                ret\_node = successor

        if not ret\_node:

            return

            # 需要更新height

        ret\_node.height = 1 + max(

            self.get\_height(ret\_node.left),

            self.get\_height(ret\_node.right),

        )

        banlance\_factor = self.get\_balance\_factor(ret\_node)

        #LL

        if banlance\_factor > 1 and self.get\_balance\_factor(ret\_node.left) >= 0:

            return self.right\_rotate(ret\_node)

        #RR

        if banlance\_factor < -1 and self.get\_balance\_factor(ret\_node.right) <= 0:

            return self.left\_rotate(ret\_node)

        # LR

        if banlance\_factor > 1 and self.get\_balance\_factor(ret\_node.left) < 0:

            ret\_node.left = self.left\_rotate(ret\_node.left)

            return self.right\_rotate(ret\_node)

        # RL

        if banlance\_factor < -1 and self.get\_balance\_factor(ret\_node.right) > 0:

            ret\_node.right = self.right\_rotate(ret\_node.right)

            return self.left\_rotate(ret\_node)

        return ret\_node

### 3.3.2 Film.py

class film():

    def \_\_init\_\_(self,name,director,actors,types,rating,region,language,time,plot,url,click\_times=0,review='noreview'):

        #filmname,director,actors,types,ratings,regions,languages,date,intro

    #def \_\_init\_\_(self,name,director,types):

        self.name = name

        self.rating = rating

        #self.adapt\_from = adapt\_from

        self.region = region.split('/')

        self.types = types.split('/')

        self.plot = plot

        self.director = director

        self.actors = actors.split('/')

        self.language = language.split('/')

        self.time = time

        #self.types = types

        self.click\_times = int(click\_times)

        self.image='./img/'+self.name+'.webp'

        self.url=url

        self.review=review.split('/')

    def clicked(self):

        self.click\_times = self.click\_times + 1

### 3.3.3 GUI.py

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox

from PyQt5 import QtCore

from Ui\_GUI import \*

from Ui\_review import \*

from search import Search

import webbrowser

class MyQLabel(QtWidgets.QLabel):

    button\_clicked\_signal = QtCore.pyqtSignal()

    def \_\_init\_\_(self, parent=None):

        super(MyQLabel, self).\_\_init\_\_(parent)

        self.node=None

    def mouseReleaseEvent(self, QMouseEvent):

        self.button\_clicked\_signal.emit()

    def connect\_customized\_slot(self, func):

        self.button\_clicked\_signal.connect(func)

class Review(QtWidgets.QMainWindow, Ui\_review):

    def \_\_init\_\_(self, parent=None):

        super(Review, self).\_\_init\_\_(parent)

        self.setupUi(self)

        self.review=''

    def ConfirmClicked(self):

        self.review=self.Enter.toPlainText()

        self.my\_Signal.emit(self.review)

        self.close()

    def CancelClicked(self):

        self.close()

    my\_Signal = QtCore.pyqtSignal(str)

class MyWindow(QMainWindow, Ui\_FSS):

    def \_\_init\_\_(self, parent=None):

        super(MyWindow, self).\_\_init\_\_(parent)

        self.s=Search()

        self.setFocusPolicy(QtCore.Qt.StrongFocus)

        self.rev=Review()

        self.setFocus()

#        self.SearchButton.clicked.connect(self.SearchButtonClicked)

        self.setupUi(self)

        self.lastimg=None

        self.allhead=None

        self.init\_pages()

        self.sym=1

        self.current\_url=''

        self.currentfilm=None

    my\_Signal = QtCore.pyqtSignal(str)

    def eventFilter(self, obj, event):

        #if obj == self.SearchText:

        if event.type() == QtCore.QEvent.KeyPress and event.key() == QtCore.Qt.Key\_Enter:

            self.SearchButtonClicked()

            return True

        return QtCore.QObject.eventFilter(self, obj, event)

    def SearchButtonClicked(self):

        strs=self.s.search(self.SearchText.text())

        if strs=='No Such Film!':

            self.imformation.setText(strs)

            QMessageBox.question(self, '错误',"未找到符合条件的电影",QMessageBox.Ok)

            return

        #self.imformation.setText(strs)

        self.allhead=strs.head

        self.lastimg=strs.head

        pointer=self.lastimg

        self.sym=4

        self.img1.node=pointer

        if pointer is not None:

            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img2.node=pointer

        if pointer is not None:

            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img3.node=pointer

        if pointer is not None:

            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img4.node=pointer

        if pointer is not None:

            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img5.node=pointer

        if pointer is not None:

            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img6.node=pointer

        if pointer is not None:

            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img7.node=pointer

        if pointer is not None:

            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=pointer.next

        else:

            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

        if pointer is not None:

            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

        else:

            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img8.node=pointer

        if pointer is not None:

            self.lastimg=pointer.next

    def DeleteButtonClicked(self):

        if self.s.avl.contains(self.SearchText.text()):

            reply=QMessageBox.question(self, '提示',"您确定要删除吗?", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

            if reply==QMessageBox.Yes:

                self.s.delete(self.SearchText.text())

                QMessageBox.question(self, '成功',"已删除",QMessageBox.Ok)

        else:

            self.imformation.setText('No Such Film!')

            QMessageBox.question(self, '错误',"未找到符合条件的电影",QMessageBox.Ok)

            return

    def init\_pages(self):

        pointer=self.s.rating\_sort.head.filmlist.head

        self.allhead=pointer

        self.img1.node=pointer

        if pointer is not None:

            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img2.node=pointer

        if pointer is not None:

            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img3.node=pointer

        if pointer is not None:

            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img4.node=pointer

        if pointer is not None:

            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img5.node=pointer

        if pointer is not None:

            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img6.node=pointer

        if pointer is not None:

            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img7.node=pointer

        if pointer is not None:

            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

        if pointer is not None:

            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

        else:

            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img8.node=pointer

        if pointer is not None:

            self.lastimg=self.s.rating\_sort.gnext(pointer)

    def NextPageClicked(self):

        pointer=self.lastimg

        self.img1.node=pointer

        if self.sym == 1:

            if pointer is not None:

                self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img2.node=pointer

            if pointer is not None:

                self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img3.node=pointer

            if pointer is not None:

                self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img4.node=pointer

            if pointer is not None:

                self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img5.node=pointer

            if pointer is not None:

                self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img6.node=pointer

            if pointer is not None:

                self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img7.node=pointer

            if pointer is not None:

                self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.rating\_sort.gnext(pointer)

            else:

                self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

            if pointer is not None:

                self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

            else:

                self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img8.node=pointer

            if pointer is not None:

                self.lastimg=self.s.rating\_sort.gnext(pointer)

        elif self.sym==2:

            if pointer is not None:

                self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img2.node=pointer

            if pointer is not None:

                self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img3.node=pointer

            if pointer is not None:

                self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img4.node=pointer

            if pointer is not None:

                self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img5.node=pointer

            if pointer is not None:

                self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img6.node=pointer

            if pointer is not None:

                self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img7.node=pointer

            if pointer is not None:

                self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=self.s.clicks\_sort.gnext(pointer)

            else:

                self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

            if pointer is not None:

                self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

            else:

                self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img8.node=pointer

            if pointer is not None:

                self.lastimg=self.s.clicks\_sort.gnext(pointer)

        elif self.sym == 4:

            if pointer is not None:

                self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img2.node=pointer

            if pointer is not None:

                self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img3.node=pointer

            if pointer is not None:

                self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img4.node=pointer

            if pointer is not None:

                self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img5.node=pointer

            if pointer is not None:

                self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img6.node=pointer

            if pointer is not None:

                self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img7.node=pointer

            if pointer is not None:

                self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                pointer=pointer.next

            else:

                self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

            if pointer is not None:

                self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

            else:

                self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

            self.img8.node=pointer

            if pointer is not None:

                self.lastimg=pointer.next

    def Show\_film(self):

        send = self.sender()

        self.currentfilm=send.node.film

        if send.node is not None:

            send.node.film.clicked()

            self.s.refresh\_list\_external(self.s.clicks\_sort,send.node.film)

            \_translate = QtCore.QCoreApplication.translate

            self.current\_url=send.node.film.url

            if send.node.film.review[0]=='noreview':

                self.imformation.setHtml('电影名称:'+send.node.film.name+'<br />'+\

                    '导演:'+send.node.film.director+'<br />'+\

                    '演员:'+' '.join(send.node.film.actors)+'<br />'+\

                    '类型:'+' '.join(send.node.film.types)+'<br />'+\

                    '上映日期: '+send.node.film.time+'<br />'+\

                    '语言:'+' '.join(send.node.film.language)+'<br />'+\

                    '区域:'+' '.join(send.node.film.region)+'<br />'+\

                    '评分:'+send.node.film.rating+'<br />'+\

                    '剧情梗概:'+send.node.film.plot+'<br />'+\

                    '<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><a href=\"'+\

                    send.node.film.url+\

                    '\"><span style=\" text-decoration: underline; color:#0000ff;\">'+\

                    send.node.film.url+'</span></a></p>')

            else:

                self.imformation.setHtml('电影名称:'+send.node.film.name+'<br />'+\

                    '导演:'+send.node.film.director+'<br />'+\

                    '演员:'+' '.join(send.node.film.actors)+'<br />'+\

                    '类型:'+' '.join(send.node.film.types)+'<br />'+\

                    '上映日期: '+send.node.film.time+'<br />'+\

                    '语言:'+' '.join(send.node.film.language)+'<br />'+\

                    '区域:'+' '.join(send.node.film.region)+'<br />'+\

                    '评分:'+send.node.film.rating+'<br />'+\

                    '剧情梗概:'+send.node.film.plot+'<br />'+\

                    '影评:<br />'+'<br />'.join(send.node.film.review)+'<br />'+\

                    '<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><a href=\"'+\

                    send.node.film.url+\

                    '\"><span style=\" text-decoration: underline; color:#0000ff;\">'+\

                    send.node.film.url+'</span></a></p>')

        else:

            #self.imformation.setText('error')

            pass

    def FrontPageClicked(self):

        pointer=self.allhead

        if pointer!=self.lastimg:

            tmp1=pointer

            tmp=pointer

            while pointer is not None:

                tmp1=tmp

                tmp=pointer

                if self.sym == 1:

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.rating\_sort.gnext(pointer)

                    if pointer == self.lastimg:

                        pointer=tmp1

                        if pointer == None:

                            pointer=tmp

                        if self.img1.node == self.lastimg:

                            pointer=tmp

                        self.img1.node=pointer

                        if pointer is not None:

                            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img2.node=pointer

                        if pointer is not None:

                            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img3.node=pointer

                        if pointer is not None:

                            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img4.node=pointer

                        if pointer is not None:

                            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img5.node=pointer

                        if pointer is not None:

                            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img6.node=pointer

                        if pointer is not None:

                            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img7.node=pointer

                        if pointer is not None:

                            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.rating\_sort.gnext(pointer)

                        else:

                            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        if pointer is not None:

                            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

                        else:

                            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img8.node=pointer

                        if pointer is not None:

                            self.lastimg=self.s.rating\_sort.gnext(pointer)

                        break

                elif self.sym == 2:

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer is not None:

                        pointer=self.s.clicks\_sort.gnext(pointer)

                    if pointer == self.lastimg:

                        pointer=tmp1

                        if pointer == None:

                            pointer=tmp

                        if self.img1.node == self.lastimg:

                            pointer=tmp

                        self.img1.node=pointer

                        if pointer is not None:

                            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img2.node=pointer

                        if pointer is not None:

                            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img3.node=pointer

                        if pointer is not None:

                            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img4.node=pointer

                        if pointer is not None:

                            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img5.node=pointer

                        if pointer is not None:

                            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img6.node=pointer

                        if pointer is not None:

                            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img7.node=pointer

                        if pointer is not None:

                            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=self.s.clicks\_sort.gnext(pointer)

                        else:

                            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        if pointer is not None:

                            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

                        else:

                            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img8.node=pointer

                        if pointer is not None:

                            self.lastimg=self.s.clicks\_sort.gnext(pointer)

                        break

                elif self.sym == 4:

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer is not None:

                        pointer=pointer.next

                    if pointer == self.lastimg:

                        pointer=tmp1

                        if pointer == None:

                            pointer=tmp

                        if self.img1.node == self.lastimg:

                            pointer=tmp

                        self.img1.node=pointer

                        if pointer is not None:

                            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img2.node=pointer

                        if pointer is not None:

                            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img3.node=pointer

                        if pointer is not None:

                            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img4.node=pointer

                        if pointer is not None:

                            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img5.node=pointer

                        if pointer is not None:

                            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img6.node=pointer

                        if pointer is not None:

                            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img7.node=pointer

                        if pointer is not None:

                            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

                            pointer=pointer.next

                        else:

                            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        if pointer is not None:

                            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

                        else:

                            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

                        self.img8.node=pointer

                        if pointer is not None:

                            self.lastimg=pointer.next

                        break

    def DirectorButtonClicked(self):

        strs=self.s.search('directors')

        self.imformation.setText(strs)

    def ActorButtonClicked(self):

        strs=self.s.search('actors')

        self.imformation.setText(strs)

    def ListButtonClicked(self):

        strs=self.s.search('list')

        self.imformation.setText(strs)

    def TypesButtonClicked(self):

        strs=self.s.search('types')

        self.imformation.setText(strs)

    def RatingButtonClicked(self):

        pointer=self.s.rating\_sort.head.filmlist.head

        self.allhead=pointer

        self.img1.node=pointer

        self.sym=1

        if pointer is not None:

            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img2.node=pointer

        if pointer is not None:

            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img3.node=pointer

        if pointer is not None:

            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img4.node=pointer

        if pointer is not None:

            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img5.node=pointer

        if pointer is not None:

            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img6.node=pointer

        if pointer is not None:

            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img7.node=pointer

        if pointer is not None:

            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.rating\_sort.gnext(pointer)

        else:

            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

        if pointer is not None:

            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

        else:

            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img8.node=pointer

        if pointer is not None:

            self.lastimg=self.s.rating\_sort.gnext(pointer)

    def HeatButtonClicked(self):

        pointer=self.s.clicks\_sort.head.filmlist.head

        self.allhead=pointer

        self.img1.node=pointer

        self.sym=2

        if pointer is not None:

            self.img1.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img1.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img2.node=pointer

        if pointer is not None:

            self.img2.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img2.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img3.node=pointer

        if pointer is not None:

            self.img3.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img3.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img4.node=pointer

        if pointer is not None:

            self.img4.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img4.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img5.node=pointer

        if pointer is not None:

            self.img5.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img5.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img6.node=pointer

        if pointer is not None:

            self.img6.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img6.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img7.node=pointer

        if pointer is not None:

            self.img7.setPixmap(QtGui.QPixmap(pointer.film.image))

            pointer=self.s.clicks\_sort.gnext(pointer)

        else:

            self.img7.setPixmap(QtGui.QPixmap('./img/timg.png'))

        if pointer is not None:

            self.img8.setPixmap(QtGui.QPixmap(pointer.film.image))

        else:

            self.img8.setPixmap(QtGui.QPixmap('./img/timg.png'))

        self.img8.node=pointer

        if pointer is not None:

            self.lastimg=self.s.clicks\_sort.gnext(pointer)

    def openlinks(self):

        webbrowser.open(self.current\_url)

    def save\_file(self):

        self.s.avl.preorder\_traversal\_save()

    def Show\_review(self):

        self.rev=Review()

        self.rev.my\_Signal.connect(self.AddReview)

        self.rev.show()

    def AddReview(self):

        if self.currentfilm != None:

            if self.currentfilm.review[0]=='noreview':

                self.currentfilm.review=[]

                self.currentfilm.review.append(self.rev.Enter.toPlainText())

            else:

                self.currentfilm.review.append(self.rev.Enter.toPlainText())

    def closeEvent(self, QCloseEvent):

        choice = QMessageBox.question(self, '提示', '您确定要关闭吗？', QMessageBox.Yes | QMessageBox.No)

        if choice == QMessageBox.Yes:

            self.save\_file()

            QCloseEvent.accept()

        elif choice == QMessageBox.No:

            QCloseEvent.ignore()

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    myWin = MyWindow()

    myWin.show()

    sys.exit(app.exec\_())

1.4

from Film import film

class linked\_list():

    class node():

        def \_\_init\_\_(self,film):

            self.film = film

            self.next = None

            self.strs=''

    def \_\_init\_\_(self):

        self.head = None

    def \_append(self,film,node):

        if self.head == None:

            self.head = self.node(film)

        elif node.next is not None:

            self.\_append(film,node.next)

        else:

            node.next = self.node(film)

    def append(self,film):

        self.\_append(film,self.head)

    def is\_empty(self):

        if self.head is None:

            return True

        else:

            return False

    def delete(self,film\_name):

        p=self.head

        q=self.head

        if p is not None:

            if p.film.name == film\_name:

                self.head=p.next

                return

            q=q.next

            while q is not None:

                if q.film.name == film\_name:

                    p.next=q.next

                    break

                q=q.next

                p=p.next

    def \_show(self,node):

        if node is not None:

            print(node.film.name,end=' ')

            self.strs=self.strs+'   '+node.film.name

            self.\_show(node.next)

    def show(self):

        self.strs=''

        self.\_show(self.head)

        return self.strs

    def lenth(self):

        i=0

        p=self.head

        while p!=None:

            p=p.next

            i=i+1

        return i

### 3.3.4 linkedlist.Py

from Film import film

class linked\_list():

    class node():

        def \_\_init\_\_(self,film):

            self.film = film

            self.next = None

            self.strs=''

    def \_\_init\_\_(self):

        self.head = None

    def \_append(self,film,node):

        if self.head == None:

            self.head = self.node(film)

        elif node.next is not None:

            self.\_append(film,node.next)

        else:

            node.next = self.node(film)

    def append(self,film):

        self.\_append(film,self.head)

    def is\_empty(self):

        if self.head is None:

            return True

        else:

            return False

    def delete(self,film\_name):

        p=self.head

        q=self.head

        if p is not None:

            if p.film.name == film\_name:

                self.head=p.next

                return

            q=q.next

            while q is not None:

                if q.film.name == film\_name:

                    p.next=q.next

                    break

                q=q.next

                p=p.next

    def \_show(self,node):

        if node is not None:

            print(node.film.name,end=' ')

            self.strs=self.strs+'   '+node.film.name

            self.\_show(node.next)

    def show(self):

        self.strs=''

        self.\_show(self.head)

        return self.strs

    def lenth(self):

        i=0

        p=self.head

        while p!=None:

            p=p.next

            i=i+1

        return i

### 3.3.5 sortredlinkedlist.py

from linkedlist import linked\_list

from Film import film

class sorted\_linked\_list():

    class node():

        def \_\_init\_\_(self,weights):

            self.weights=weights

            self.filmlist=linked\_list()

            self.next=None

            self.strs=''

            self.symbol=0

    def \_\_init\_\_(self):

        self.head=None

    def gnext(self,node):

        if node.next is not None:

            return node.next

        elif self.symbol == 1:

            if float(node.film.rating) == float(self.head.weights):

                if self.head.next is not None:

                    return self.head.next.filmlist.head

            else:

                p=self.head.next

                while p is not None:

                    if float(node.film.rating) == float(p.weights):

                        if p.next is not None:

                            return p.next.filmlist.head

                    p=p.next

        elif self.symbol == 2:

            if float(node.film.click\_times) == float(self.head.weights):

                if self.head.next is not None:

                    return self.head.next.filmlist.head

            else:

                p=self.head.next

                while p is not None:

                    if float(node.film.click\_times) == float(p.weights):

                        if p.next is not None:

                            return p.next.filmlist.head

                    p=p.next

        return None

    def insert\_by\_rating(self,film):

        self.symbol=1

        if self.head is None:

            self.head=self.node(float(film.rating))

            self.head.filmlist.append(film)

        else:

            p=self.head

            q=self.head

            while(float(p.weights)>float(film.rating)):

                q=p

                p=p.next

                if p is None:

                    break

            if q==p:

                if(p.weights!=film.rating):

                    self.head=self.node(film.rating)

                    self.head.filmlist.append(film)

                    self.head.next=p

                    return

            if p is not None:

                if float(p.weights)<float(film.rating):

                    q.next=self.node(float(film.rating))

                    q.next.filmlist.append(film)

                    q.next.next=p

                else:

                    p.filmlist.append(film)

            else:

                q.next=self.node(float(film.rating))

                q.next.filmlist.append(film)

                q.next.next=p

    def insert\_by\_clicks(self,film):

        self.symbol=2

        if self.head is None:

            self.head=self.node(film.click\_times)

            self.head.filmlist.append(film)

        else:

            p=self.head

            q=self.head

            while(p.weights>film.click\_times):

                q=p

                p=p.next

                if p is None:

                    break

            if q==p:

                if(p.weights!=film.click\_times):

                    self.head=self.node(film.click\_times)

                    self.head.filmlist.append(film)

                    self.head.next=p

                    return

            if p is not None:

                if p.weights<film.click\_times:

                    q.next=self.node(film.click\_times)

                    q.next.filmlist.append(film)

                    q.next.next=p

                else:

                    p.filmlist.append(film)

            else:

                q.next=self.node(film.click\_times)

                q.next.filmlist.append(film)

                q.next.next=p

    def \_show(self,node):

        if node is not None:

            if node.weights != 0:

                print('\n'+str(node.weights)+':',end="")

                self.strs=self.strs+str(node.weights)+':\n'+node.filmlist.show()+'\n'

                self.\_show(node.next)

    def show(self):

        self.strs=''

        self.\_show(self.head)

        print('')

        return self.strs

    def \_delete(self,film\_name,node):

        if node is not None:

            node.filmlist.delete(film\_name)

            self.\_delete(film\_name,node.next)

    def delete\_null\_node(self):

        p=self.head

        q=self.head

        if p is not None:

            if p.filmlist.is\_empty():

                self.head=p.next

                return

            q=q.next

            while q is not None:

                if q.filmlist.is\_empty():

                    p.next=q.next

                    break

                q=q.next

                p=p.next

    def delete(self,film\_name):

        self.\_delete(film\_name,self.head)

        self.delete\_null\_node()

if \_\_name\_\_=='\_\_main\_\_':

    class test():

        def \_\_init\_\_(self,rating):

            self.rating = rating

### 3.3.6 Search.py

from Film import film

from AVLTree import AVLTree

from sortedlinkedlist import sorted\_linked\_list

import re

class Search():

    def \_\_init\_\_(self):

        self.all\_types = []

        self.directors = []

        self.actors = []

        self.words = ''

        self.strs=''

        with open('filmlist.txt', 'r',encoding='utf-8') as f:

            self.words = f.read()

        self.words = self.words.split()

        self.avl=AVLTree()

        self.rating\_sort=sorted\_linked\_list()

        self.clicks\_sort=sorted\_linked\_list()

        for word in self.words:

            wd = word.split('#')

            if self.avl.contains(wd[0]):

                print("duplicated")

            else:

                import\_film=film(wd[0],wd[1],wd[2],wd[3],wd[4],wd[5],wd[6],wd[7],wd[8],wd[9],wd[10],wd[11])

                self.avl.add\_from\_root(import\_film)#filmname,director,actors,types,ratings,regions,languages,date,intro

                self.rating\_sort.insert\_by\_rating(import\_film)

                self.clicks\_sort.insert\_by\_clicks(import\_film)

                tp=wd[3].split('/')

                actorslist=wd[2].split('/')

                directorslist=wd[1].split('/')

                for typ in tp:

                    if self.all\_types.count(typ) < 1:

                        self.all\_types.append(typ)

                for actor in actorslist:

                    if self.actors.count(actor) < 1:

                        self.actors.append(actor)

                for director in directorslist:

                    if self.directors.count(director) < 1:

                        self.directors.append(director)

    def refresh\_list(self,linkedlist, films):

        for film in films:

            linkedlist.delete(film.film\_name)

            linkedlist.insert\_by\_clicks(film.film)

    def refresh\_list\_external(self,linkedlist, film):

        linkedlist.delete(film.name)

        linkedlist.insert\_by\_clicks(film)

    def show\_types(self):

        print('There are',len(self.all\_types),'types.')

        print(self.all\_types)

        self.strs=''

        i=0

        for types in self.all\_types:

            self.strs=self.strs+types+'\t'

            i+=1

            if i==4:

                self.strs=self.strs+'\n'

                i=0

        return self.strs

    def show\_actors(self):

        print('There are',len(self.actors),'actors.')

        print(self.actors)

        self.strs=''

        i=0

        for actor in self.actors:

            self.strs=self.strs+actor+'   '

            i+=1

            if i==2:

                self.strs=self.strs+'\n'

                i=0

        return self.strs

    def show\_directors(self):

        print('There are',len(self.directors),'directors.')

        print(self.directors)

        i=0

        for director in self.directors:

            self.strs=self.strs+director+'   '

            i+=1

            if i==2:

                self.strs=self.strs+'\n'

                i=0

        return self.strs

    def list\_all(self):

        print('There are',self.avl.size,'films.')

        return self.avl.root\_preorder\_traversal()

    def search(self,search\_str):

        self.search\_word = search\_str.split()

        i = 0

        for word in self.search\_word:

            if self.all\_types.count(word) < 1 and self.directors.count(word) < 1 and self.actors.count(word) < 1:

                i = 1

        if i == 0:

            str1=self.avl.root\_preorder\_traversal\_search(search\_str)

            if len(self.avl.need\_refresh\_list)!=0:

                self.refresh\_list(self.clicks\_sort, self.avl.need\_refresh\_list)

                return str1

        elif search\_str == 'list':

            return self.list\_all()

        elif search\_str == 'types':

            return self.show\_types()

        elif search\_str == 'directors':

            return self.show\_directors()

        elif search\_str == 'actors':

            return self.show\_actors()

        elif search\_str == 'ratingsort':

            return self.rating\_sort.show()

        elif search\_str == 'heatsort':

            return self.clicks\_sort.show()

        elif re.match(r'delete .\*',search\_str):

            self.delete(search\_str.split()[1])

        else:

            #searched\_film = self.avl.get\_film\_from\_node(search\_str)

            str1=self.avl.get\_film\_from\_node\_llist(search\_str)

        if len(self.avl.need\_refresh\_list)!=0:

            self.refresh\_list(self.clicks\_sort, self.avl.need\_refresh\_list)

            return str1

        return ''

    def append(self):

        pass

    def delete(self,strs):

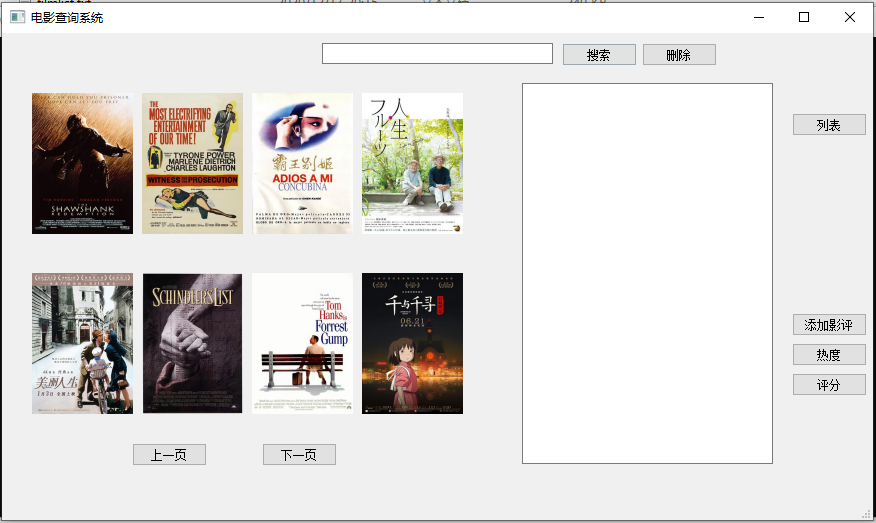
        self.avl.remove(strs)

        self.rating\_sort.delete(strs)

        self.clicks\_sort.delete(strs)

# 

# 4用户手册及测试结果

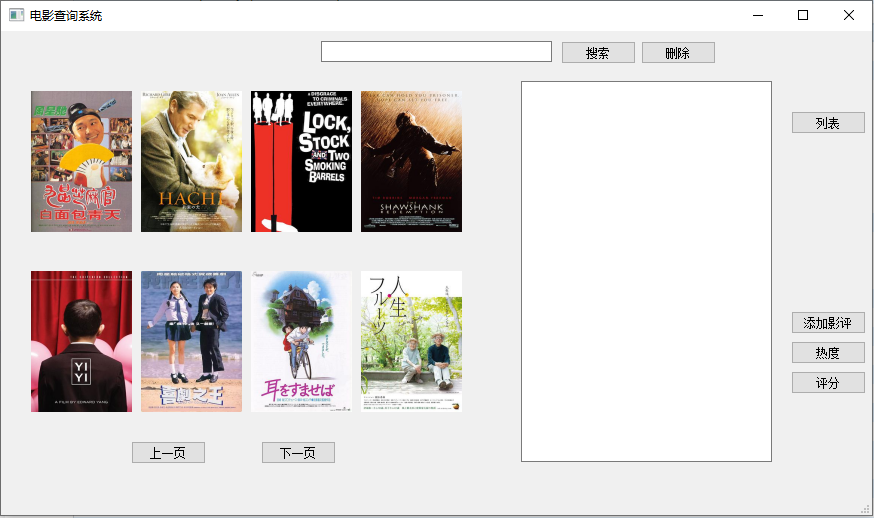


系统创建以后，可以看到共八部电影的海报，点击后可展示信息



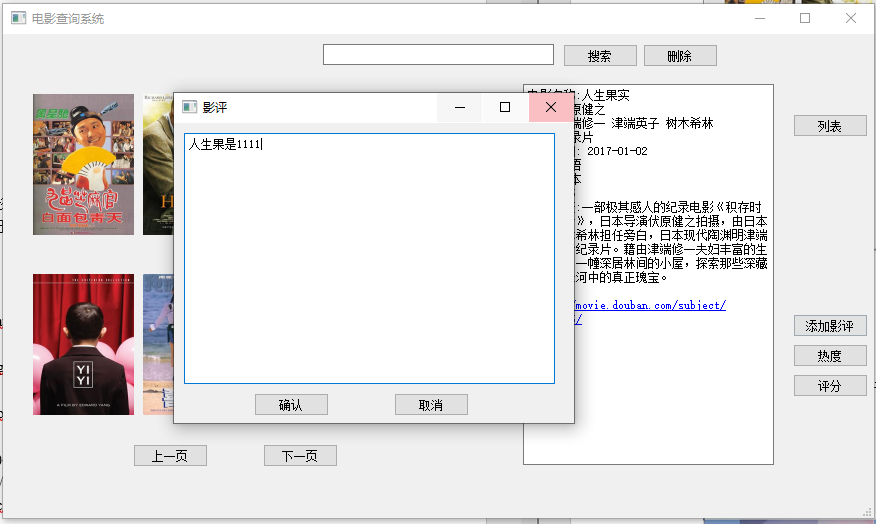
## 1、排序演示

在未选择排序时，默认为评分排序，可以在右侧点击切换排序。



## 2、添加评论

之后可以为任意电影添加评语，在右下角点击添加影评并输入评语



点击确定后



可以看到影评已成功添加

## 3、交叉查询演示

搜索爱情 剧情。



可以看到所有爱情剧情标签的电影

在进一步缩小范围，搜索爱情 剧情 周星驰



可以看到只剩下了喜剧之王

## 4、访问网页

点击电影信息最下方的链接



可以跳转到相应的网页



## 5、修改文件

电影的所有信息包括可修改的热度，影评都储存于filmlist.txt中，在程序关闭时，会将本次修改的信息重新写入文件。

# 5总结提高

同学依据自己的经历写出个性化的总结。内容可以包括：程序开发中的体会与收获，开发中遇到的问题与解决情况，自己对自己完成课设情况的评价等等。

## 5.1体会与收获

每次课设的制作都让我收获很多，他能让我重新巩固以前学习的知识，更加熟练的运用编程语言，尤其是这次，对于我来说还是有着很大的挑战的。在最开始的阶段，选择哪种数据结构、选择哪些算法让我很纠结。最后选择了树加链表的双重结构。也让我对于这两种数据结构的御用更加的娴熟。每当我解决一个问题以后，所获得的那种巨大的成就感，都会让我有信心去解决下一个问题，这将鼓励我、支持我继续走下去。

## 5.2问题与解决

一是对python的不熟悉，由于语言的不同，python对缩进的要求是非常严格的，而且与C++不同的是，在如if for的时候后面要加冒号，对语言的不熟悉大大减慢了变成的效率。同样对图形界面的不熟悉，在GUI的制作中用的是pyqt5。因此也需要付出很大的学习成本，但最后还是克服了困难并完成了。

## 5.3评价

在界面的优化上也不够到位，应该在左侧使用电影图片和名字。

# 6参考文献

1．《数据结构与算法》

2．https://www.python.org/doc/ python官方手册

3. https://www.runoob.com/python3/python3-tutorial.htmlPython 3 教程