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11月27日

1需求分析

1.1需要完成的功能

电影分类可以按豆瓣评分排行、名著改编、地域（欧美，大陆，港台，日韩）、剧情（爱情、喜剧、犯罪、动画、历史等）、某导演（至少包含你本人欣赏的3位导演（彰显个性））、某演员作品（至少包含你本人关注的3位演员）、语言（英语、汉语、日语等）等。请设计一个查询系统，通过介绍用户感兴趣电影的剧情、导演、演员表、影评、豆瓣评分等信息，让用户充分了解此部电影，供用户参考。

1. 提供分类（至少包含上述7种），用户选择一种分类，在候选影片（要求按分类查询的搜索结果至少包含6部电影）中选择查询，而且各种分类之间要求有交叉；

1.1、豆瓣评分

1.2、名著改编

1.3、地域

1.4、剧情

1.5、导演

1.6、演员

1.7、语言

2、允许用户输入电影名称，直接查询；

3、按最热（依据本系统的点击率）、最新（本系统包含影片的上映日期）排行查询。

3.1、最热排行

3.2、最新排行

【要求】

1、选择最适合的数据结构和最适合的算法实现；

2、要求只能存储原始数据（电影基本信息及其分类特征（包含多个，格式自定义）），不允许存储统计结果（各种分类是按“分类特征”进行统计的结果）；对数据统计的结果要求能够依据原始数据的改变而变化。例如，文件保存的原始数据中(例如：某影片的剧情由喜剧改为了历史，在喜剧分类中，就看不到此片)，而不是存储最终的统计结果；

3、要求“原始数据”存储在文本文件中（如果数据是爬虫爬出来的大量真实数据，可以有限度的使用数据库，但不允许使用数据库中的插删查找功能解决本问题（要求用数据结构知识解决），并需要举例展示来证明没有使用插删查找功能），格式自定义。每次运行，要求从文件中读取数据。

4、能够展示每部影片的详细信息，包括 影片简介（编剧、导演、主演、类型、制片国家、语言、上映日期、片长）剧情简介、豆瓣评分、影评等等；

5、界面展示力争做到：言简意赅，形象生动，层次越少越好，界面信息越全越好，尽量减少用户的文字的输入；

2数据结构设计

2.1主要数据结构

2.1.1电影类（Film）

class film():

    '''def \_\_init\_\_(self,rating,adapt\_from,region,plot,types,director,actors,language,time):'''

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

#构造函数，初始化电影名称，导演，类别等内容

    def clicked(*self*):

#调用click()后，clicked属性加1，位置后热度排序做准备

2.1.2AVL树类（AVLTree）

2.1.2.1节点类（film\_node）

    class film\_node:

        def \_\_init\_\_(*self*, *film\_name*=None,*director*=None,*types*=None):

            self.film\_name = film\_name

            self.left = None

            self.right = None

            self.height = 1

            self.film = film(film\_name,director,types)

*from* Film *import* film

class AVLTree:

    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 is\_empty(*self*):

#判断书是否为空

    def add\_from\_root(*self*, *film\_name*, *director*,*types*):

    def add(*self*, *node*, *film\_name*, *director*,*types*):

#添加节点

    def get\_height(*self*, *node*):

    #获取节点高度

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

    #计算平衡因子balance\_factor

    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 get\_film\_from\_node(*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*):

#前序周游

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

    def preorder\_traversal\_search(*self*,*node*,*types*):

#前序周游，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

2.1.3搜索类（Search）

*from* Film *import* film

*from* AVLTree *import* AVLTree

class Search():

    def \_\_init\_\_(*self*):

#构造函数，初始化函数，从文件读入数据，并建立avl树

    def show\_types(*self*):

#打印所有的类型

    def list\_all(*self*):

#打印所有储存的电影

    def search(*self*,*search\_str*):

#根据search\_str搜索对应的电影或对应的类型对应的电影

    def delete(*self*):

#删除

2.1.3链表类（linked\_list、sorted\_linked\_list）

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详细设计

3.1.1 Film.py

class film():

    def \_\_init\_\_(*self*,*name*,*director*,*actors*,*types*,*rating*,*region*,*language*,*time*,*plot*):

*#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 = 0

    def clicked(*self*):

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

3.1.2 AVLTree.py

*from* Film *import* film

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')

    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.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

    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.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)

    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*

*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

3.1.3search.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])

                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.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\_print(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)

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

            self.\_show(node.next)

    def show(*self*):

        self.strs=''

        self.\_show(self.head)

*return* self.strs

*if* \_\_name\_\_ == "\_\_main\_\_":

    llist=linked\_list()

    llist.append(1)

    llist.append(2)

    llist.append(3)

    llist.append(4)

    llist.append(5)

*pass*

3.1.5 sortedlinkedlist.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=''

    def \_\_init\_\_(*self*):

        self.head=None

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

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

                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<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*):

*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 is not 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

    sll=sorted\_linked\_list()

    sll.insert\_by\_rating(test('8.7'))

    sll.insert\_by\_rating(test('7.1'))

    sll.insert\_by\_rating(test('9.7'))

    sll.insert\_by\_rating(test('8.2'))

    sll.insert\_by\_rating(test('8.4'))

    sll.insert\_by\_rating(test('8.1'))

    sll.insert\_by\_rating(test('8.1'))

    sll.insert\_by\_rating(test('8.1'))

*pass*

3.1.6 Ui\_GUI.py

*from* PyQt5 *import* QtCore, QtGui, QtWidgets

class Ui\_FSS(object):

    def setupUi(*self*, *FSS*):

        FSS.setObjectName("FSS")

        FSS.resize(488, 504)

        self.centralwidget = QtWidgets.QWidget(FSS)

        self.centralwidget.setObjectName("centralwidget")

        self.SearchButton = QtWidgets.QPushButton(self.centralwidget)

        self.SearchButton.setGeometry(QtCore.QRect(310, 20, 75, 23))

        self.SearchButton.setObjectName("SearchButton")

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

        self.SearchText = QtWidgets.QLineEdit(self.centralwidget)

        self.SearchText.setGeometry(QtCore.QRect(50, 20, 231, 21))

        self.SearchText.setObjectName("SearchText")

        self.imformation = QtWidgets.QTextEdit(self.centralwidget)

        self.imformation.setGeometry(QtCore.QRect(50, 60, 331, 381))

        self.imformation.setReadOnly(True)

        self.imformation.setObjectName("imformation")

        self.DeleteButton = QtWidgets.QPushButton(self.centralwidget)

        self.DeleteButton.setGeometry(QtCore.QRect(400, 20, 75, 23))

        self.DeleteButton.setObjectName("DeleteButton")

        self.DeleteButton.clicked.connect(self.DeleteButtonClicked)

*# self.List = QtWidgets.QPushButton(self.centralwidget)*

*# self.List.setGeometry(QtCore.QRect(390, 110, 75, 23))*

*# self.List.setObjectName("List")*

*# self.List.clicked.connect(self.ListButtonClicked)*

*# self.Director = QtWidgets.QPushButton(self.centralwidget)*

*# self.Director.setGeometry(QtCore.QRect(390, 160, 75, 23))*

*# self.Director.setObjectName("Director")*

*# self.Director.clicked.connect(self.DirectorButtonClicked)*

*# self.actor = QtWidgets.QPushButton(self.centralwidget)*

*# self.actor.setGeometry(QtCore.QRect(390, 210, 75, 23))*

*# self.actor.setObjectName("actor")*

*# self.actor.clicked.connect(self.ActorButtonClicked)*

*# self.types = QtWidgets.QPushButton(self.centralwidget)*

*# self.types.setGeometry(QtCore.QRect(390, 260, 75, 23))*

*# self.types.setObjectName("types")*

*#self.types.clicked.connect(self.TypesButtonClicked)*

        self.Heat = QtWidgets.QPushButton(self.centralwidget)

        self.Heat.setGeometry(QtCore.QRect(390, 310, 75, 23))

        self.Heat.setObjectName("Heat")

        self.Heat.clicked.connect(self.HeatButtonClicked)

        self.rating = QtWidgets.QPushButton(self.centralwidget)

        self.rating.setGeometry(QtCore.QRect(390, 360, 75, 23))

        self.rating.setObjectName("rating")

        self.rating.clicked.connect(self.RatingButtonClicked)

        FSS.setCentralWidget(self.centralwidget)

        self.menubar = QtWidgets.QMenuBar(FSS)

        self.menubar.setGeometry(QtCore.QRect(0, 0, 488, 23))

        self.menubar.setObjectName("menubar")

        FSS.setMenuBar(self.menubar)

        self.statusbar = QtWidgets.QStatusBar(FSS)

        self.statusbar.setObjectName("statusbar")

        FSS.setStatusBar(self.statusbar)

        self.retranslateUi(FSS)

        QtCore.QMetaObject.connectSlotsByName(FSS)

    def retranslateUi(*self*, *FSS*):

        \_translate = QtCore.QCoreApplication.translate

        FSS.setWindowTitle(\_translate("FSS", "电影查询系统"))

        self.SearchButton.setText(\_translate("FSS", "搜索"))

        self.DeleteButton.setText(\_translate("FSS", "删除"))

*# self.List.setText(\_translate("FSS", "列表"))*

*# self.Director.setText(\_translate("FSS", "导演"))*

*# self.actor.setText(\_translate("FSS", "演员"))*

*# self.types.setText(\_translate("FSS", "类别"))*

        self.Heat.setText(\_translate("FSS", "热度"))

        self.rating.setText(\_translate("FSS", "评分"))

3.1.7 GUI.py

*import* sys

*from* PyQt5.QtWidgets *import* QApplication, QMainWindow, QMessageBox

*from* PyQt5 *import* QtCore

*from* Ui\_GUI *import* \*

*from* search *import* Search

class MyWindow(QMainWindow, Ui\_FSS):

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

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

        self.s=Search()

        self.setFocusPolicy(QtCore.Qt.StrongFocus)

        self.setFocus()

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

        self.setupUi(self)

    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)

    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 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 HeatButtonClicked(*self*):

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

        self.imformation.setText(strs)

    def RatingButtonClicked(*self*):

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

        self.imformation.setText(strs)

*if* \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    myWin = MyWindow()

    myWin.show()

    sys.exit(app.exec\_())

4总结

进一步完善了所涉及的数据结构，如avl树，链表等。对程序的逻辑结构有了一个完整的思路，略微进行了对GUI的编写，了解了PyQt是如何运行的，我受益匪浅。