**目录**

[一、 问题描述 1](#_Toc153933228)

[二、 数据结构设计 1](#_Toc153933229)

[全局变量： 1](#_Toc153933230)

[实例变量： 1](#_Toc153933231)

[数据结构： 1](#_Toc153933232)

[链接存储的线性表 2](#_Toc153933233)

[无向图表示共现矩阵： 3](#_Toc153933234)

[三元组存储节省空间： 4](#_Toc153933235)

[三、 模块与算法设计 5](#_Toc153933236)

[无序排序的加速算法Q： 5](#_Toc153933237)

[基于BERT的文本相度计算： 7](#_Toc153933238)

[四、 测试用例设计 7](#_Toc153933239)

[五、 系统运行及测试过程 9](#_Toc153933240)

[1．多个python文件： 9](#_Toc153933241)

[2．开发者路径： 10](#_Toc153933242)

[3．相似度计算的代码构建： 10](#_Toc153933243)

[4．字符串格式问题： 10](#_Toc153933244)

[5．搜索界面的位置问题： 11](#_Toc153933245)

[6．格式调整： 12](#_Toc153933246)

[7．歌曲重叠： 12](#_Toc153933247)

[8．其他问题： 13](#_Toc153933248)

[六、 程序运行示例 13](#_Toc153933249)

[运行示例： 13](#_Toc153933250)

[注意事项： 18](#_Toc153933251)

[七、 总结 19](#_Toc153933252)

[体会： 19](#_Toc153933253)

[改进： 19](#_Toc153933254)

# 问题描述

本项目为针对音乐应用层的信息管理系统。

其应用无向图实现关联共现次数的统计，使用三元组降低空间储存成本，利用无序排序的加速算法Q实现运行速度的优化，并结合了预训练的大语言模型SimBert-Base-Chinese实现了用户相似度分析和歌曲智能推荐。最后考虑了用户的个性化需求基于tinker模块进行下游可视化呈现和交互。

# 数据结构设计

## 全局变量：

全局变量有songs\_df（总曲库），accounts（所有用户的信息），system\_background（公司的logo，使用DALL·E生成）定义它们为全局变量主要是它们和示例无关，且不希望被轻易修改（特别是logo）

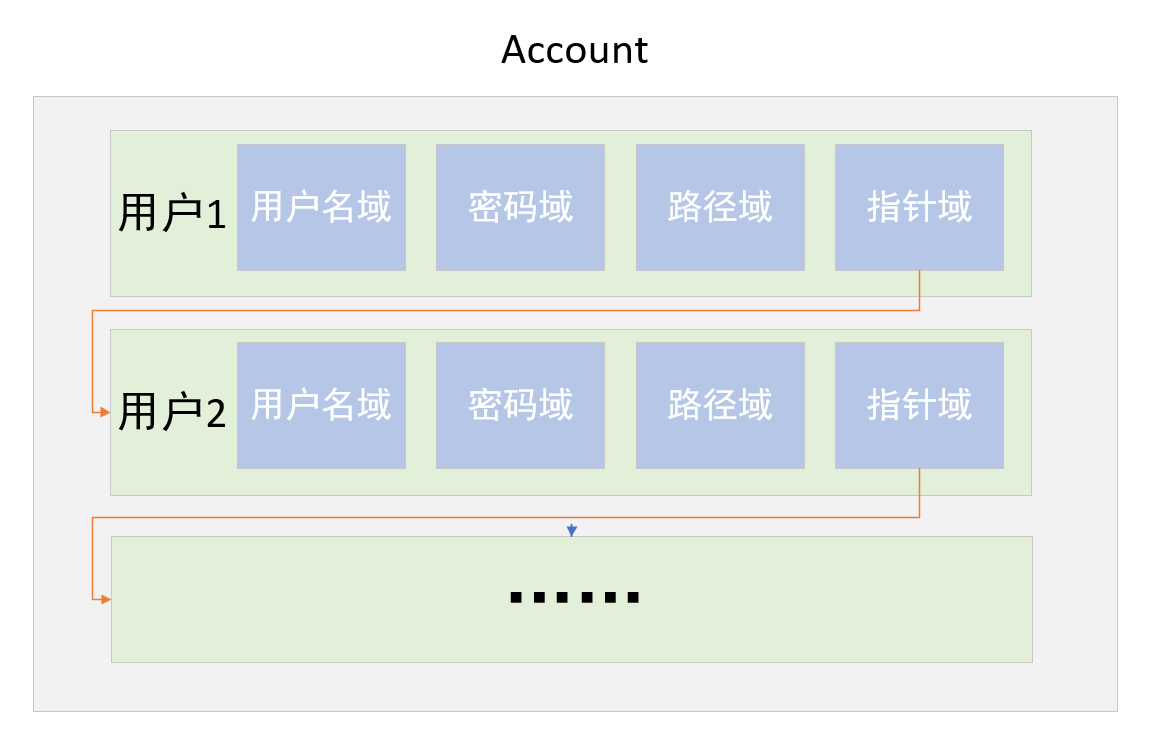
## 实例变量：

比较重要的有df（用户曲库），triplets\_list（用户曲库对应的关联图储存），及tinker的各模块，它们都是和用户实例相关的，主要是考虑到它们要在各函数中的调用

## 数据结构：

本项目主要使用的数据结构为：1. 线性表 2. 无向图 3. 三元组

### 链接存储的线性表



顺序存储的线性表的缺点是要预先指定内存，但是在信息系统里面用户的数量是很难预先估计的，因此我们用了链接存储取代了顺序存储。这有很大的实际意义，因为在应用层中，用户的数量可能会非常大，超出了预先指定的范围。

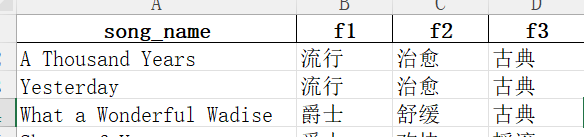
图形用户界面, Teams

描述已自动生成

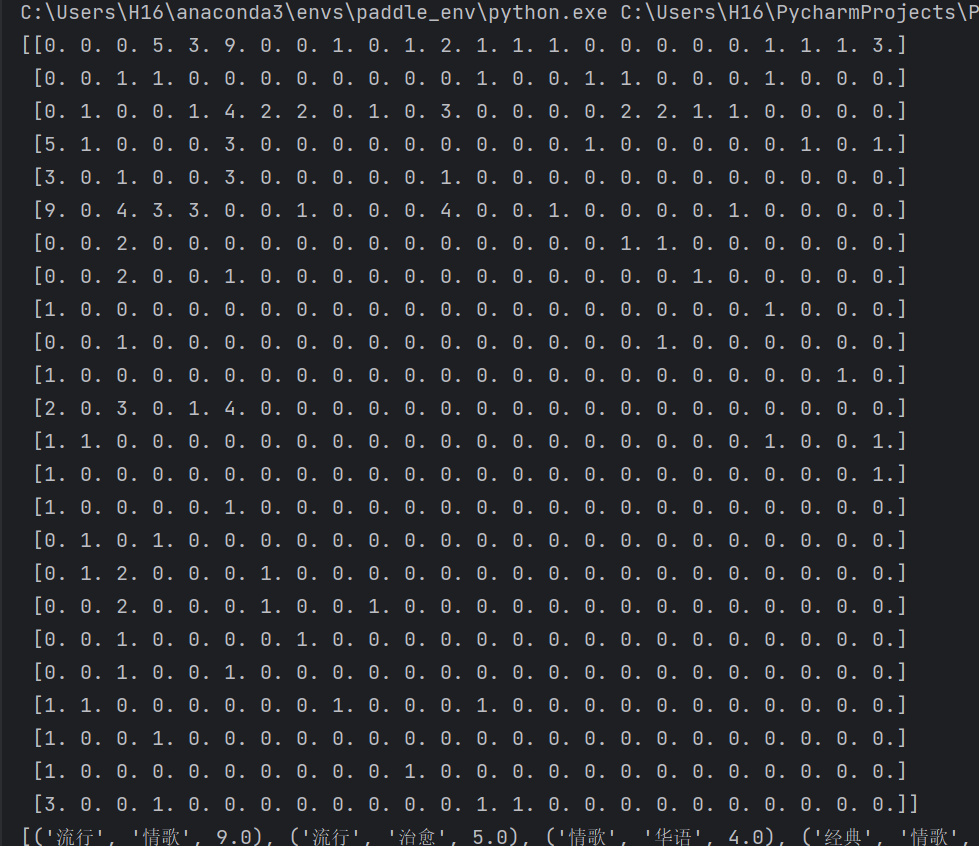
对比线性存储

### 无向图表示共现矩阵：

我们设计了用三个特征（feature）来表示一首歌的风格特征（实际应用中可以增加）



一个用户喜欢的列表里面会有很多歌，但是只是统计它们出现的次数不能体现它们的相互关系，因此我们引入了共现矩阵，以无向图表示：



然后选取了最高的五个，作为分析

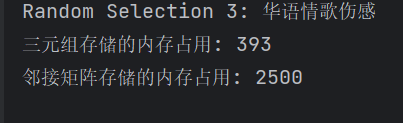
### 三元组存储节省空间：

由数据结构的可知，上图是一个稀疏矩阵，用邻接矩阵存储会更节省空间。但是由于本任务只涉及到统计共现对的次数，而不涉及某一个风格特征和其他所有风格特征的关系，因此采取了进一步的优化，用三元组降低存储空间。理论如下：

三元组的空间复杂度是O（E），但是邻接表是O（V+2E) 也就是O（V+E）。

实现了很可观的提升，去查找的时候也会更快。

内存占用的实际情况如下：



# 模块与算法设计

本节介绍本项目采用的两个算法：

### 无序排序的加速算法Q：

本项目排序算法未掉包，自己实现了上述算法。为了验证，我初始化了一个元素为100的不重复乱序正整数列表，下面是该算法和原有的排序的时间比对：



可以看到优化的时间在一个数量级以上，其中Not Found的情况接近两个数量级。在项目中源码如下：屏幕上有字

描述已自动生成

### 基于BERT的文本相度计算：

本人研究的是深度学习方向的，此项目的核心功能推荐算法的相似度计算采用了自然语言处理大模型的方法。由于和数据结构关系较弱，简单的采用了百度开源paddlepaddle（飞桨）里面的Taskflow模块，基于里面的simbert-base-chinese预训练模型。下面简单介绍一下：

BERT是一个在Transformer的encoder架构基础上演化来的大预言模型，它延申了一系列的模型，如ALBERT。虽现在采用基于Transformer的decoder架构的GPT家族为主用于文本生成任务，但是在文本理解上，BERT架构由于其中的自注意力模块没有单向掩码，有更强的能力。本模型是其在中文语料库上训练的简化模型。

效果简单演示：



# 测试用例设计

构建了一些虚拟的人p1, p2, p3，并填充了一些数据：

图形用户界面, 文本, 应用程序

描述已自动生成

表格

描述已自动生成

P1的部分数据

文本

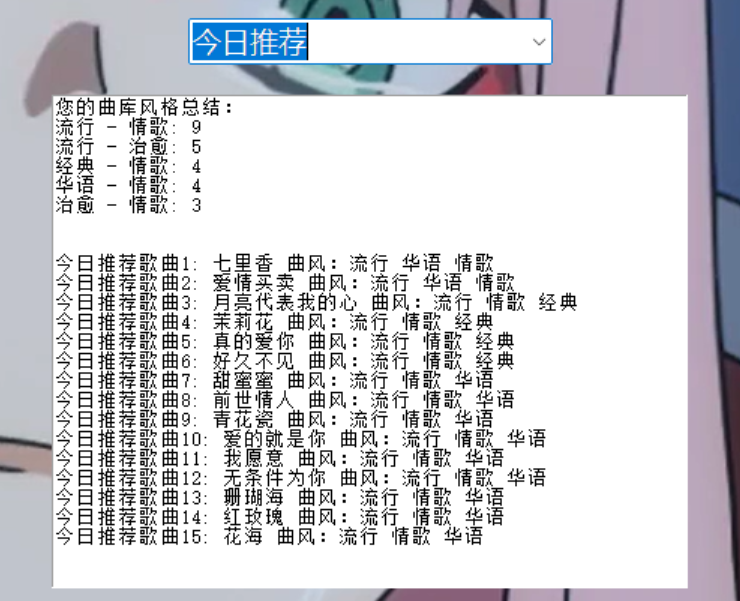
描述已自动生成

测试我的喜欢功能

文本

描述已自动生成

Debug可见具体的每一条的文本相似度



歌曲均不在P1的歌单中，且曲风与其相近

各模块测试结果均一切正常

# 系统运行及测试过程

## 1．多个python文件：

全程的代码调试一直基于PyCharm，是因为它方便的debug功能。测试我和主程序一起构建了好多个python文件

文本

描述已自动生成

分布是recommend\_graph、recommend\_triplets和sim，第一个是最初写的用图来表示共现关系，在课程代码的基础上修改。然后我比对了空间占用，写了第二个三元组储存；Sim文件是用来调试paddle环境和分模块测试文本相似度运行的。

## 2．开发者路径：

其中，因为每次输入用户名和密码非常麻烦，我设置了开发者路径：



无需用户名和密码，便于调试，接下来讲述代码书写过程中的一些历程和debug的经验。

## 3．相似度计算的代码构建：

文本

描述已自动生成

sim = similarity([[f, top\_features\_str]])[0]["similarity"]这一行代码的[0] ["similarity"]是采用debug的方式我查看similarity对象的属性得出的

电脑萤幕的截图

描述已自动生成

## 4．字符串格式问题：

在文本处理的过程中也采用了很多的debug来查看str的形式：

文本

描述已自动生成

## 5．搜索界面的位置问题：

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 网站

描述已自动生成

经检查发现是它的插入位置有问题，这里不能填master，而是要放在main\_frame里面

## 6．格式调整：

手机屏幕截图

描述已自动生成

这里的换行最开始也有问题，即全部挤在一起了。上图是修正完的结果（错的没截屏下来）

## 7．歌曲重叠：

还有一个主要的bug是数据库里面的歌曲有重叠，而且因为推荐的歌曲要在听过的个人曲库之外，我采取了先把个人曲库和总曲库进行concat操作再drop\_duplicates的操作，这时候若同一首歌被编码了多个特征

如：

表格

描述已自动生成

那去重的时候这两个就不会被去除，因为drop\_duplicates方法指定了整行相同。

我比对了merged\_df，unique\_df，df，song\_df的元素数量，发现了这个问题，最后我重新整理了数据库，解决了这个问题。

## 8．其他问题：

当然debug的时候还遇到了变量命名等等细节问题，如song和name\_song，这主要是和小组成员交流代码是时候导致的，本文不再赘述。

# 程序运行示例

## 运行示例：

本项目已在github上开源：[Halloweenpink/-music-information-manage-system: 来自本人结构科学大作业，基于paddlenlp实现用户歌曲推荐 (github.com)](https://github.com/Halloweenpink/-music-information-manage-system)



登录界面

输入用户名和密码登录

图形用户界面

描述已自动生成

主界面（自定义用户背景）

图片包含 图形用户界面

描述已自动生成

两张功能

图形用户界面

描述已自动生成

查看我的喜欢

图形用户界面, 应用程序

描述已自动生成

搜索曲库

图形用户界面

描述已自动生成

添加我的喜欢（会直接作用于用户数据库）

图形用户界面

描述已自动生成

删除歌曲（会直接作用于用户数据库）

文本, QR 代码

描述已自动生成

系统自动计算推荐曲目（来自非用户听过的歌曲）

## 注意事项：

考虑用户需求，初始化窗口会自动全屏调整为最大；

若在python环境内运行，数据存放结构如下：

----music\_project

----all\_person

----person\_name1.xlsx

----person\_name2.xlsx

----person\_name3.xlsx

----……

----0.png

----1.png

----data.py

----音乐信息系统.py

其中0.png和1.png为默认的登录背景与用户背景的图片命名

Paddle下载链接：<https://www.paddlepaddle.org.cn/install/quick>

本项目采用gpu版，若部署电脑没有独立显卡，可下载cpu版本

Paddle版本：paddlepaddle-gpu 2.5.2.post116

Cuda版本：11.6

paddlenlp版本：2.6.1

python版本：3.8

# 总结

## 体会：

1. 在项目实现中，发现tinker模块不熟，调整了好久
2. 切实的感觉到了数据结构的重要性，空间优化和运行速度是一个算法工程师应该考虑的事情
3. 下游的部署很重要
4. 之前大一也写过很长的代码，但是没有这么多面向对象的封装，有了函数定义、命名，再去调用的时候确实会清楚不少

## 改进：

1. 其实可以把歌名一起输入nlp模型进行文本分析的，考虑到只是一个示例项目这里跳过了，只使用了数据挖掘出的最高共现项
2. 数据来源于基于gpt4的自监督训练，我们人为设定了一些歌曲和歌曲的风格，让大语言模型补全了数据。若爬取个人的实际听音记录效果会更好
3. 项目在运行的时候每次都要加载一次内存里面的模型，这是主要的卡顿时间，若部署在服务器上应该可以交互的更快，但是我还没学这个
4. 可以考虑使用图神经网络来进行分析，而不是只用BERT，或者采用最前沿的轻量化encoder架构试一下。推荐系统其实和图挺相关的，时间所限未完成