**第一章 引言**

**1.项目内容**

本项目主要实现了基于关键词的站内搜索，并通过PageRank算法对搜索结果排名。

**2.基本需求**

开发的爬虫能爬取与关键词相关的网页，并且对该网页展开广度优先遍历。能解决编码问题，能够分析网页含有的关键词，并对关键词及其对应的URL保存至MySQL数据库，为以后的搜索提供数据支持。

**第二章 概要设计**

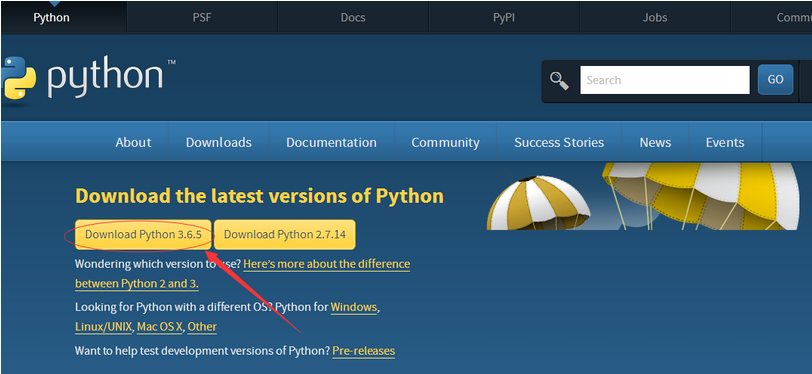
**1.平台**

本项目采用Python语言编写，其中引入了jieba、numpy、urllib、pymysql、BeautifulSoup等Python库。MySQL数据库作为存储介质。PageRank算法把搜索结果按URL的权重排序。

**2.搭建过程**

**2.1.搭建Python的运行环境**

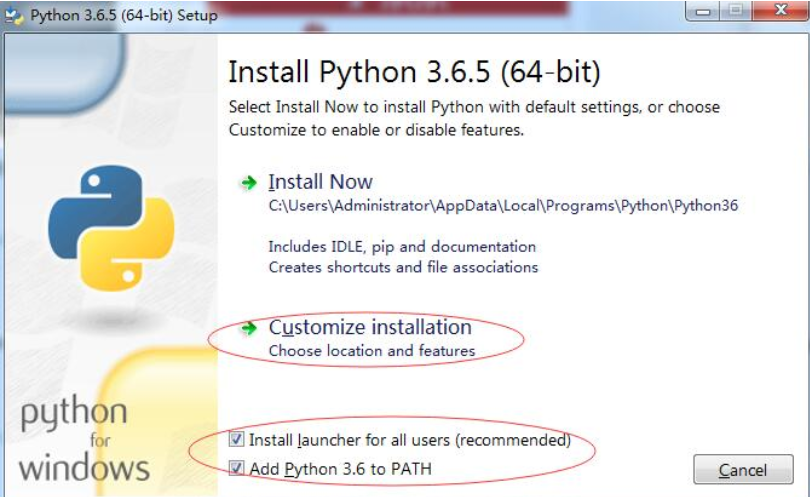
访问Python官网https://www.python.org/downloads/下载Python。

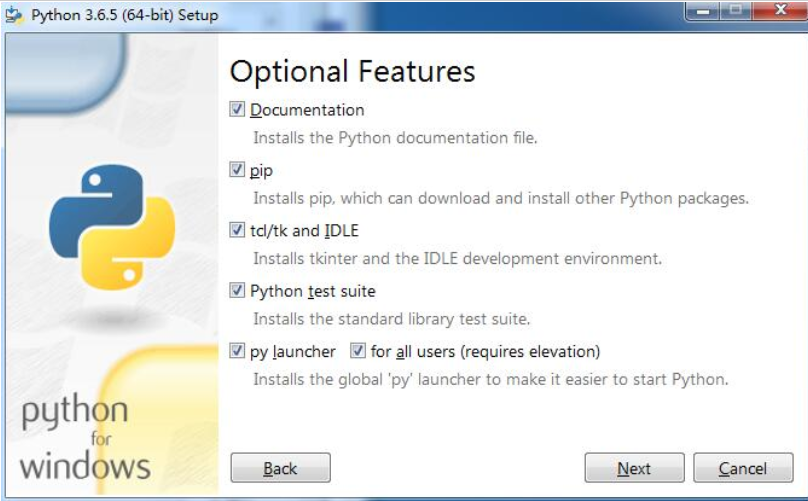


Python安装步骤如下：

1）双击下载的python3.6.5.exe文件

2）在1）的界面勾选Install launcher for all users(recommended)和Add Python 3.6 to PATH， 点击Customize installation,跳转到如下界面，点击Next跳转至Advance Option，点击Brower，选择文件夹。





3）点击Install。至此安装结束

4）下面开始安装爬虫所需要的库。命令为pip install request（或BeautifulSoup或jieba或numpy或pymysql）。

**2.2.搭建MySQL数据库。**

访问MySQL官网下载压缩包。安装过程如下：

1. 解压zip包到安装目录

比如我的安装目录为C:\Sofeware\MySQL

1. 配置文件

在安装根目录下添加 my.ini，写入基本配置，如下：

[mysqld]

# 设置3306端口

port=3306

# 设置mysql的安装目录

basedir=C:\\Sofeware\\MySQL\\mysql-8.0.11-winx64

# 设置mysql数据库的数据的存放目录

datadir=C:\\Sofeware\\MySQL\\MySQLData

# 允许最大连接数

max\_connections=200

# 允许连接失败的次数。这是为了防止有人从该主机试图攻击数据库系统

max\_connect\_errors=10

# 服务端使用的字符集默认为UTF8

character-set-server=utf8

# 创建新表时将使用的默认存储引擎

default-storage-engine=INNODB

# 默认使用“mysql\_native\_password”插件认证

default\_authentication\_plugin=mysql\_native\_password

[mysqld\_safe]

# 定义mysql应该支持的sql语法，数据校验

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

[mysql]

# 设置mysql客户端默认字符集

default-character-set=utf8

[client]

# 设置mysql客户端连接服务端时默认使用的端口

port=3306

default-character-set=utf8

1. 初始化数据库

在MySQL安装目录的 bin 目录下执行命令：mysqld --initialize –console。执行完成后，会打印 root 用户的初始默认密码。

1. 安装服务

在MySQL安装目录的 bin 目录下执行命令：

mysqld –install

安装完成之后，通过命令net start mysql启动MySQL的服务。

1. 更改密码

在MySQL安装目录的 bin 目录下执行命令：

mysql -u root -p

这时候会提示输入密码，记住了上面第3.1步安装时的密码，填入即可登录功，进入MySQL命令模式。在MySQL中执行命令：

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_nativ

e\_password BY 'msi;123';

即可完成修改密码。

**2.3.第三方框架和插件**

Jieba库用以解决中文分词问题；BeautifulSoup用以解决提取HTML或XML文件数据的问题；urllib库提供了一系列用于操作URL的功能；numpy用来存储和处理大型矩阵，以此来解决PageRank算法中的矩阵运算问题；pymysql是一个MySQL客户端库，以此来解决Python与数据库的连接。

**2.4.功能及设计**

1）爬取站点所有的URL链接：首先选取种子URL，然后将其放入URL列表，从URL列表中选取待抓取的URL并将其下载，存储至数据库，并将此URL从URL列表中删除。以此循环上述步骤，直至URL列表为空。

2）提取网页中的中文并分词：BeautifulSoup库对网页分析，从中提取文本信息，并对网页内容进行过滤，排除非中文字符的干扰。利用jieba库对中文分词，并将其保存至一个临时列表中，为数据库的存储提供数据。

3）中文词组与其对应的URL保存至数据库，为搜索提供数据：将网页内容进行过滤，以得到中文，在对中文进行jieba分词，并将每个分词与对应的URL存至数据库。

4）搜索与关键词相关的URL链接：查询数据库，将与关键词匹配的记录返回

5）对URL链接进行排序：确定URL之间的联系，并将此种联系转化成矩阵，用PageRank算法对URL链接排序

**第三章 详细设计**

**1.爬取站点所有的URL链接**

**1.1.网络爬虫的基本工作流程**

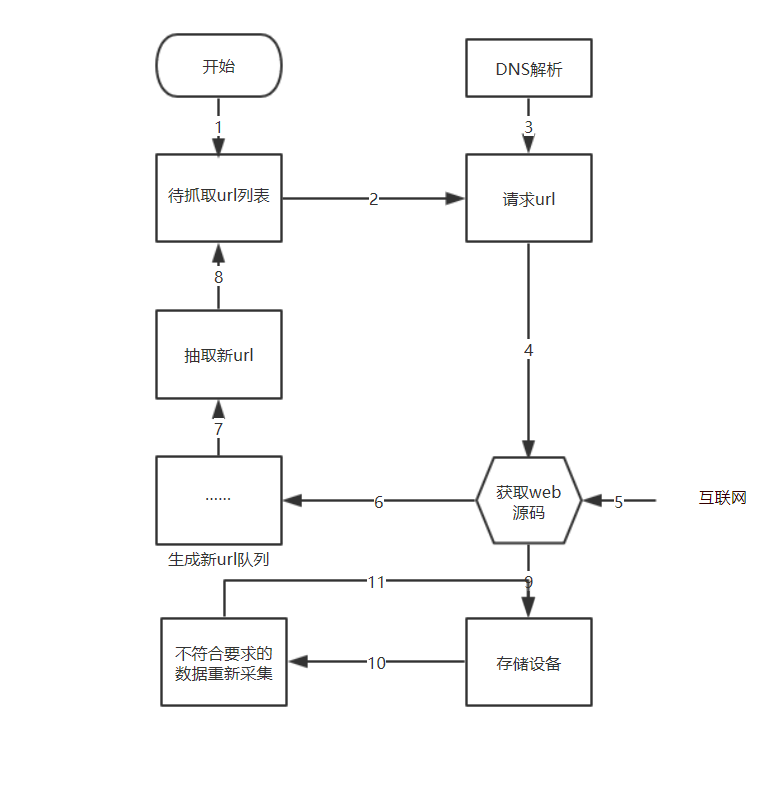
1）.首先选取种子URL（本项目中以<http://www.sdust.edu.cn/>为例）；

2）.将这些URL放入待抓取URL列表；

3）.从待抓取URL列表中取出待抓取在URL，解析DNS，并且得到主机的ip，并将URL对应的网页下载下来，存储进已下载网页库中。此外，将这些URL放进已抓取URL队列。

4）.分析已抓取URL队列中的URL，分析其中的其他URL，并且将URL放入待抓取URL队列，从而进入下一个循环。

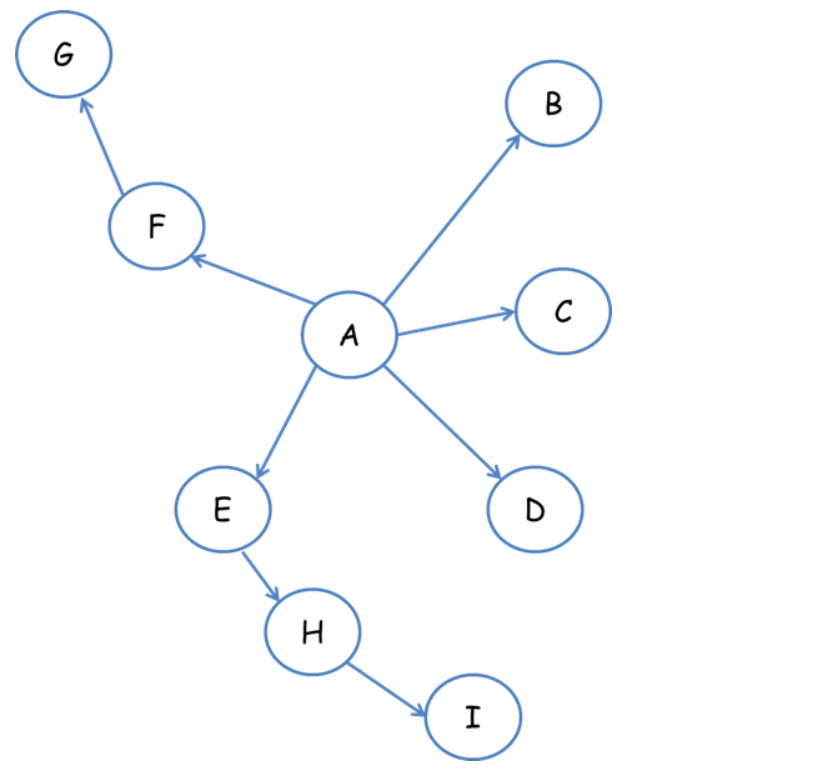
网络爬虫框架如下图所示：

****

注：存储设备为MySQL数据库

**1.2. 抓取策略**

  在爬虫系统中，待抓取URL列表是很重要的一部分。待抓取URL队列中的URL以什么样的顺序排列也是一个很重要的问题，因为这涉及到先抓取那个页面，后抓取哪个页面。而决定这些URL排列顺序的方法，叫做抓取策略。



常见的抓取策略有以下几种：

深度优先遍历策略（遍历路径：A-F-G  E-H-I B C D）

广度优先遍历策略（遍历路径：A-B-C-D-E-F G H I）

反向链接数策略

Partial PageRank策略

OPIC策略策略

本项目采用广度优先遍历策略。

代码如下：

# -\*- coding: utf-8 -\*-

import urllib.request

import urllib.parse

import urllib.error

from bs4 import BeautifulSoup

from urllib.parse import urljoin

import jieba

import re

ignorewords=set(['的','但是','然而','能','在','以及','可以','使'])

class crawler:

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

pass

def \_\_del\_\_(self):

pass

def validword(self,word):

zhPattern = re.compile(u'[\u4e00-\u9fa5]+')

match = zhPattern.search(word)

if match:

return 1

else:

return 0

def addtoindex(self,url,soup):

#if self.isindexed(url): return

print('Indexing '+url)

text=self.gettextonly(soup) #获取文本信息

words=self.separatewords(text) #分词

dicn=list(words)

title = []

for i in range(len(dicn)):

word=dicn[i]

if word in ignorewords: continue

if self.validword(word):

title.append(word)

Databases.save(title, url) #关键词和URL存库

#从一个Html网页中提取文字（不带标签的）

def gettextonly(self,soup):

v=soup.string

if v==None:

c=soup.contents

resulttext=''

for t in c: #获取文本信息

resulttext+=self.gettextonly(t)+'\n'

return resulttext

else:

return v.strip()

def separatewords(self,text): #结巴分词

seg\_list=jieba.cut\_for\_search(text);

return seg\_list

'''def separatewords(self,text):

##该方法仅仅局限于分英文单词

splitter=re.compile('\\W\*')

return [s.lower() for s in splitter.split(text) if s!=' ']

'''

#从一组小网页开始进行广度优先搜索，直到某一给定深度，期间为网页建立#索引

def crawl(self,pages,depth=2):

for i in range(depth):

newpages=set()

for page in pages: #广度

try:

c=urllib.request.urlopen(page)

except:

print ("Could not open %s" % page)

continue

soup=BeautifulSoup(c.read())

self.addtoindex(page,soup) #索引

links=soup('a')

for link in links:

if('href' in dict(link.attrs)):

url=urljoin(page,link['href'])

if url.find("'")!=-1:continue

url=url.split('#')[0] #去锚点

if url[0:4]=='http':

newpages.add(url)

pages=newpages

**2.提取网页中的中文并分词**

中文分词指的是将一个汉字序列切分成一个个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。

由于请求的网页中含有大量的英文字母、标点符号和中文。所以我们首先提取网页中的中文字符。中文字符的提取采用正则表达式的方法，代码见上述代码的函数validword(self,word)。

中文分词时需要忽略一些语气词、代词等无明确含义的字符。

jieba 基于[Python](http://lib.csdn.net/base/python" \o "Python知识库" \t "_blank)的中文分词工具,安装使用非常方便,直接pip即可

jieba.cut以及 jieba.cut\_for\_search 返回的结构都是一个可迭代的 generator

def separatewords(self,text): #结巴分词

seg\_list=jieba.cut\_for\_search(text);

return seg\_list

我们编写函数separatewords(self,text) ，调用jieba库 ，达到分词目的

当文档是英文时，也可以编写函数适应不同网站

def separatewords(self,text):

##该方法仅仅局限于分英文单词

splitter=re.compile('\\W\*')

return [s.lower() for s in splitter.split(text) if s!=' ']

**3.保存到本地链表测试**

我们开始是使用本地存储，创建动态表来存储链接

创建3个链表类型，3个节点类型(括号中表示)。

Linklist(Node):对每一个网页分词后，将词加入此链表

Weblist(Web):把网页按照所拥有的词加入词链表，接在词的后面。

Resultlist(Result):搜索结果加入此链表

class Node(object): #节点结构体

def \_\_init\_\_(self, val,wh,p=0):

self.data = val #val为string类型

self.webhead = wh

self.next = p

class LinkList(object):#链表

class Web(object): #节点结构体

class WebList(object):#链表

class Result(object):

class ResultList(object):#链表

最后输入搜索词汇，分词处理，在Linklist链表遍历，并且进行去重处理。

pagelist=['http://www.sdust.edu.cn/']

cra=crawler('')

cra.crawl(pagelist)

search = input("Please putin your keyword: ")

rl = ResultList()

rl.initlist('CONGCONG')

#seach进行结巴分词

seg\_list = jieba.cut\_for\_search(search)

for sword in seg\_list:

k = ll.index(sword)

if k != -1:

p = ll.getwh(k).getlength()

for m in range(0,p):

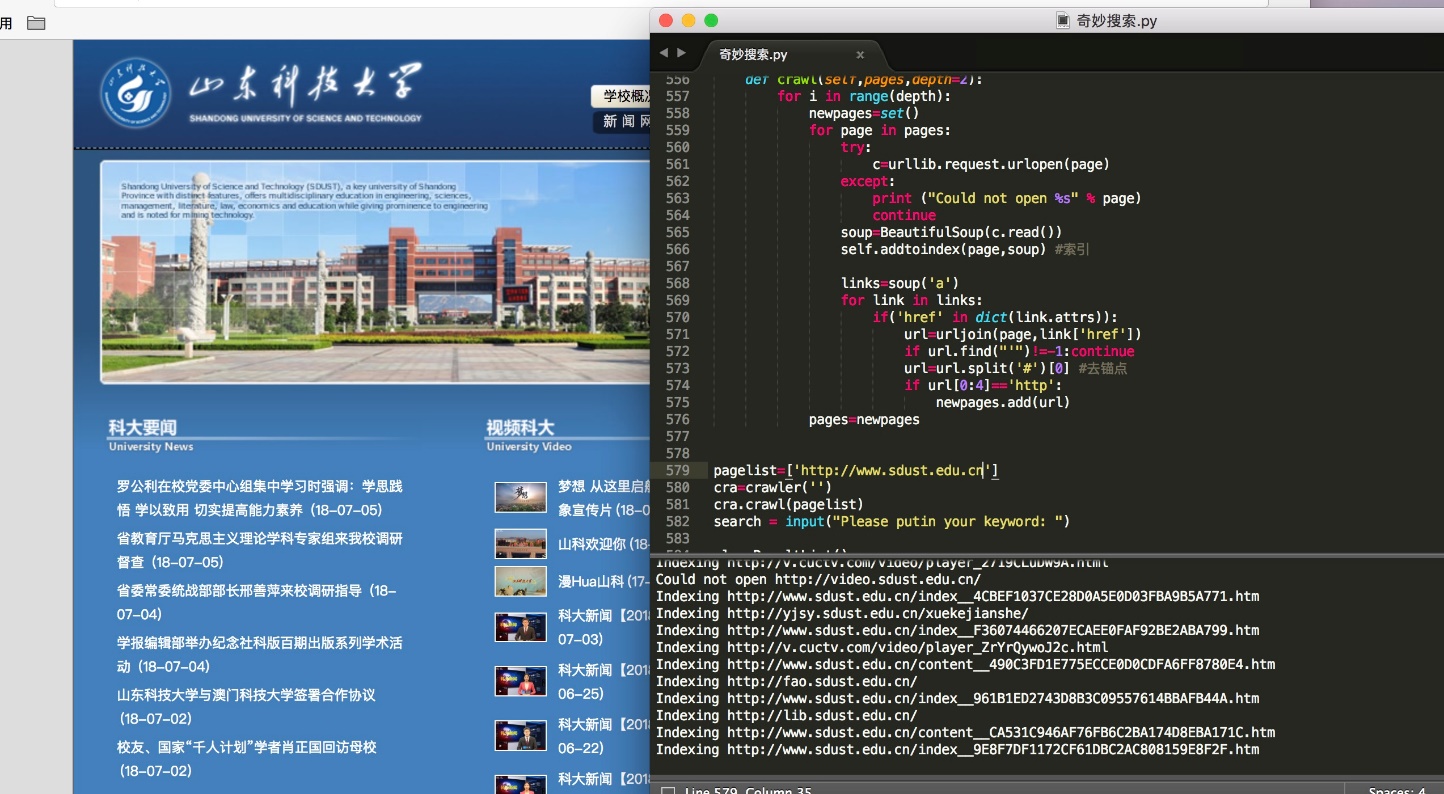
b = ll.getwh(k).getdata(m)

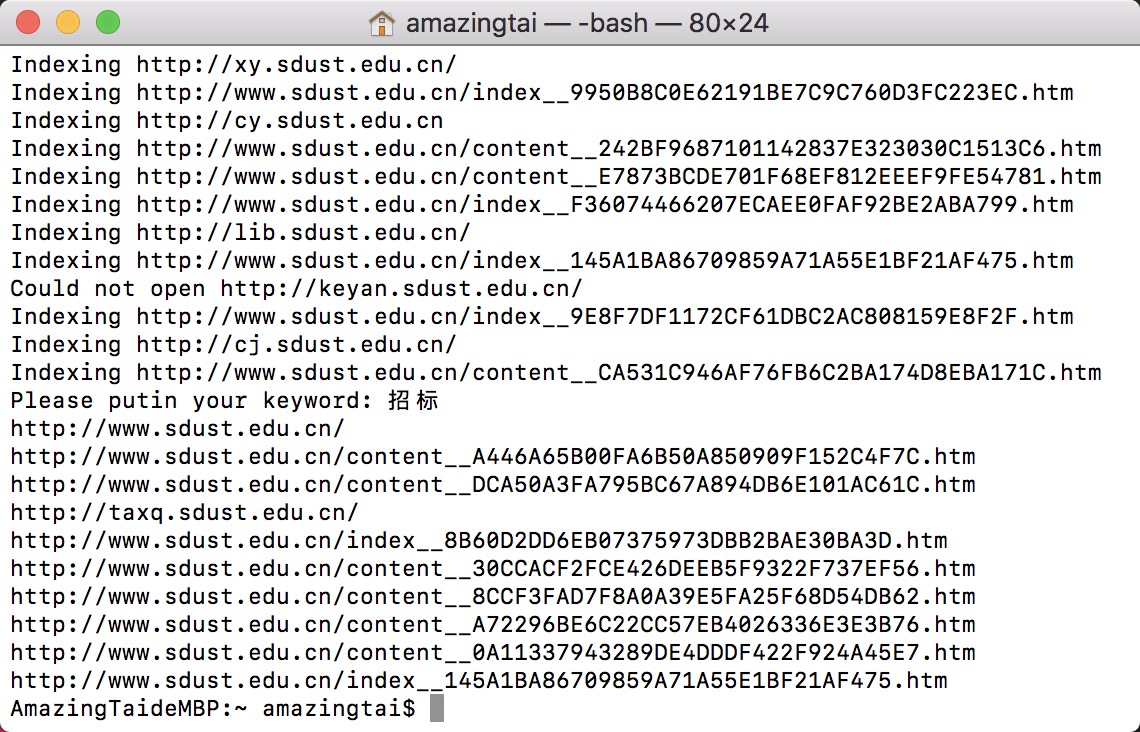
if rl.index(b) == -1:

rl.append(b)

print(b)

代码效果如下：





**3.中文词组与其对应的URL保存至数据库，为搜索提供数据**

得到的中文分词组成一个列表title，及其对应的URL链接，作为参数传给数据库文件Databases.py中的函数save(title, url)，此函数的主要功能为将title和url保存至MySQL数据库，即保存一个网页内的所有中文分词所在的URL链接。

代码如下：

def save(title, url):

# 打开数据库连接

db = pymysql.connect("localhost", "root", "msi;123", "test")

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# 使用 execute() 方法执行 SQL，如果表存在则删除

#cursor.execute("DROP TABLE IF EXISTS RESULT")

#创建表

cursor.execute("""CREATE TABLE IF NOT EXISTS RESULT

(ID INT UNSIGNED NOT NULL AUTO\_INCREMENT, TITLE CHAR(20),

URL CHAR(200), PRIMARY KEY(ID))""")

# SQL 插入语句

for each in title:

sql = "INSERT INTO RESULT(TITLE, URL)\

VALUES ('%s', '%s')" %\

(each, url)

try:

# 执行sql语句

cursor.execute(sql)

# 提交到数据库执行

db.commit()

except:

# 如果发生错误则回滚

db.rollback()

# 关闭数据库连接

db.close()

函数empty()主要功能为判断当前数据库中result表是否为空，如果为空，则返回1；否则，返回0.

代码如下：

def empty():

# 打开数据库连接

db = pymysql.connect("localhost", "root", "msi;123", "test")

# 使用cursor()方法获取操作游标

cursor = db.cursor()

try:

cursor.execute("SELECT URL FROM RESULT WHERE id = 1 ")

# 获取所有记录列表

results = cursor.fetchall()

if results:

return 0

else:

return 1

except:

return 1

**4.搜索与关键词相关的URL链接**

输入需要查询的关键词后，首先判断数据库中是否有记录，如何有记录，即数据库不为空，则直接调用数据库中记录，否则，调用crawler类中的函数crawl(pagelist)爬取pagelist中所有链接。

调用数据库文件Databases.py中的函数read(title)。该函数从MySQL数据库中查询所有与title匹配的记录，并返回列表urls。

代码如下：

def read(title):

# 打开数据库连接

db = pymysql.connect("localhost", "root", "msi;123", "test")

# 使用cursor()方法获取操作游标

cursor = db.cursor()

#sql 查询

try:

cursor.execute("SELECT URL FROM RESULT \

WHERE TITLE = '%s'" % title )

# 获取所有记录列表

results = cursor.fetchall()

#print (results)

urls = []

for row in results:

for col in row:

urls.append(col)

if urls:

return urls

else:

return 0

except:

print("Read Error")

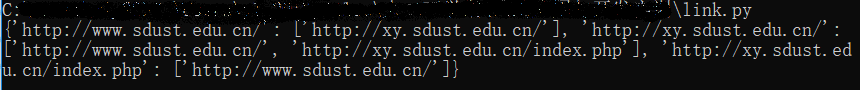
return 0

# 关闭数据库连接

db.close()

**5.对URL链接进行排序**

文件link.py实现的主要功能为将列表URL之间的可访问关系转化成字典类型的数据。函数is\_link(mainurl, securl)判断能否从mainurl 直接访问securl，如果能访问，则返回1，否则，返回0。函数linkDic(urls)主要是将列表urls转成字典。例如：

****

代码如下：

#link.py

# -\*- coding: utf-8 -\*-

from bs4 import BeautifulSoup

import urllib.request

import urllib.parse

import urllib.error

from urllib.parse import urljoin

def is\_link(mainurl, securl):

newurl = set()

try:

c=urllib.request.urlopen(mainurl)

soup=BeautifulSoup(c.read(), 'html.parser')

links=soup('a')

for link in links:

if('href' in dict(link.attrs)):

url=urljoin(mainurl,link['href'])

if url.find("'")!=-1:continue

url=url.split('#')[0] #去锚点

if url[0:4]=='http':

newurl.add(url)

if securl in newurl:

return 1

else:

return 0

except:

return 0

def linkDic(urls):

urldict = dict()

for i in urls:

turls = []

for j in urls:

if i != j:

if is\_link(i, j):

turls.append(j)

if turls:

urldict[i] = turls

else:

urldict[i] = []

#print(len(urldict))

#print(urldict)

return urldict

if \_\_name\_\_ == '\_\_main\_\_':

urls = ["http://www.sdust.edu.cn/", "http://xy.sdust.edu.cn/", "http://xy.sdust.edu.cn/

index.php?m=Index&a=articlelist&cate\_id=16",

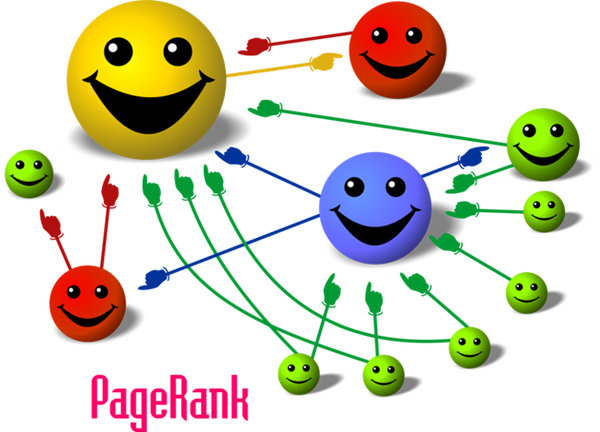
"http://xy.sdust.edu.cn/index.php?m=Index&a=article&id=2235"]

print(linkDic(urls))

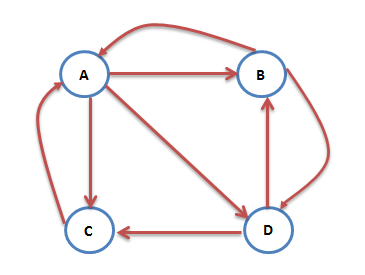
PageRank.py实现了字典型的数据转化为矩阵形式，并且实现了PageRank算法对网页的排序。数据形式的转化由函数Generate\_Transfer\_Matrix(G)实现，函数Rank(urls)提供了初始化数据，函数PageRank(M, alpha, root, node2index, index2node)实现网页的权重排序。

PageRank算法的核心思想：在互联网上，如果一个网页被很多其它网页所链接，说明它受到普遍的承认和信赖，那么它的排名就高。比如说，对来自不同网页的链接区别对待，因为那些排名高的网页的链接更可靠，于是给这些链接较大的权重。

概念图如下：



互联网中的众多网页可以看作一个有向图。下图是一个简单的例子



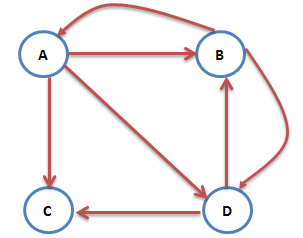
这时A的PR值就可以表示为

PR(A)=PR(B)+PR(C)

然而图中除了C之外，B和D都不止有一条出链，所以上面的计算式并不准确。想象一个用户现在在浏览B网页，那么下一步他打开A网页还是D网页在统计上应该是相同概率的。所以A的PR值应该表述为：

PR(A)=PR(B)/2 + PR(C)/1

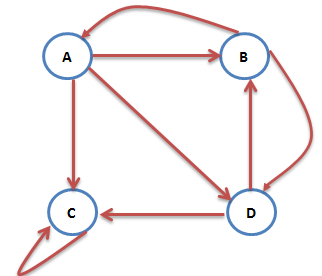
互联网中不乏一些没有出链的网页,假设图中的C网页没有出链，如下



对其他网页没有PR值的贡献，我们不喜欢这种自私的网页（其实是为了满足 Markov 链的收敛性），于是设定其对所有的网页（包括它自己）都有出链，则此图中A的PR值可表示为：

PR(A)=PR(B)/2 + PR(C)/4

然而我们再考虑一种情况：互联网中一个网页只有对自己的出链，或者几个网页的出链形成一个循环圈。那么在不断地迭代过程中，这一个或几个网页的PR值将只增不减，显然不合理。如下图中的C网页就是刚刚说的只有对自己的出链的网页。



为了解决这个问题。我们想象一个随机浏览网页的人，当他到达C网页后，显然不会傻傻地一直被C网页的小把戏困住。我们假定他有一个确定的概率会输入网址直接跳转到一个随机的网页，并且跳转到每个网页的概率是一样的。于是则此图中A的PR值可表示为：

PR(A)=α(PR(B)/2)+(1−α)/4PR(A)=α(PR(B)/2)+(1−α)/4

在一般情况下，一个网页的PR值计算如下：

PR(pi)=α∑pj∈MpiPR(pj)/L(pj)+(1−α)/N

其中Mpi是所有对pipi网页有出链的网页集合， L(pj)是网页pj的出链数目，N是网页总数，α一般取0.85。 根据上面的公式，我们可以计算每个网页的PR值，在不断迭代趋于平稳的时候，即为最终结果。具体怎样算是趋于平稳，我们在下面的PR值计算方法部分再做解释。

PageRank的计算方法：

假定向量

B=

为第一、第二、…第N个网页的网页排名。矩阵

A =

为网页之间链接的数目，其中代表第m个网页指向第n个网页的链接数。A是已知的。A是已知的，B是未知的，是我们需要计算的。

假定是第i次迭代的结果，那么

=A （1.1）

初始假设：所有的网页排名都是1/N，即

=( , , … , )

显然通过（1.1）简单的矩阵运算（但是计算量非常大）的矩阵运算，可以得到,,…。可以证明最终会收敛，即最终会无限趋于B，此时：B=B。因此，当两次迭代的结果和之间的差异非常小，接近于零时，停止迭代运算，算法结束。一般来讲，只要十次左右的迭代基本上就收敛了。

由于网页之间链接的数量相比互联网的规模非常稀疏，因此计算网页的网页排名也需要对零概率或者小概率事件进行平滑处理。网页的排名是个一维向量，对它的平滑处理只能利用一个小的常数。这时，公式（1.1）变成

=[I+(1-)A]

其中N是互联网网页的数量，是一个较小的常数，I是单位矩阵。网页排名的计算主要是矩阵相乘。

代码如下：

# coding:utf-8

import numpy as np

import link

num\_iter = 100

num\_candidates = 100 #output 100 url

def Rank(urls):

alpha = 0.85

G = link.linkDic(urls)

M, node2index, index2node = Generate\_Transfer\_Matrix(G)

result = PageRank(M, alpha, urls[0], node2index, index2node)

#print(result)

return result

def PageRank(M, alpha, root, node2index, index2node):

"""

Personal Rank in matrix formation

:param M: transfer probability matrix

:param index2node: index2node dictionary

:param node2index: node2index dictionary

:return:type of list of tuple, ex.

[(node1, prob1),(node2, prob2),...]

"""

result = []

n = len(M)

v = np.zeros(n)

#print(node2index[root])

v[node2index[root]] = 1

while np.sum(abs(v - (alpha\*np.matmul(M,v) + (1-alpha)\*v))) > 0.0001:

v = alpha \* np.matmul(M, v) + (1-alpha)\*v

for ind, prob in enumerate(v):

result.append((index2node[ind], prob))

result = sorted(result, key=lambda x:x[1], reverse=True)[:num\_candidates]

return result

def Generate\_Transfer\_Matrix(G):

"""generate transfer matrix given graph"""

index2node = dict()

node2index = dict()

for index,node in enumerate(G.keys()):

node2index[node] = index

index2node[index] = node

# num of nodes

n = len(node2index)

# generate Transfer probability matrix M, shape of (n,n)

M = np.zeros([n,n])

for node1 in G.keys():

for node2 in G[node1]:

# FIXME: some nodes not in the Graphs.keys, may incur some errors

try:

M[node2index[node2],node2index[node1]] = 1/len(G[node1])

except:

continue

return M, node2index, index2node

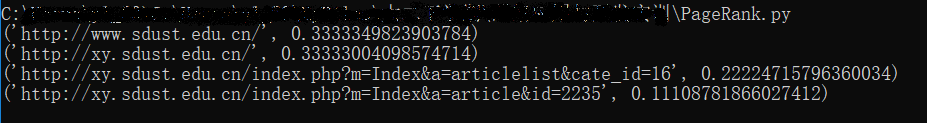
if \_\_name\_\_ == '\_\_main\_\_':

urls = ["http://www.sdust.edu.cn/", "http://xy.sdust.edu.cn/", "http://xy.sd

ust.edu.cn/index.php?m=Index&a=articlelist&cate\_id=16",

"http://xy.sdust.edu.cn/index.php?m=Index&a=article&id=2235"]

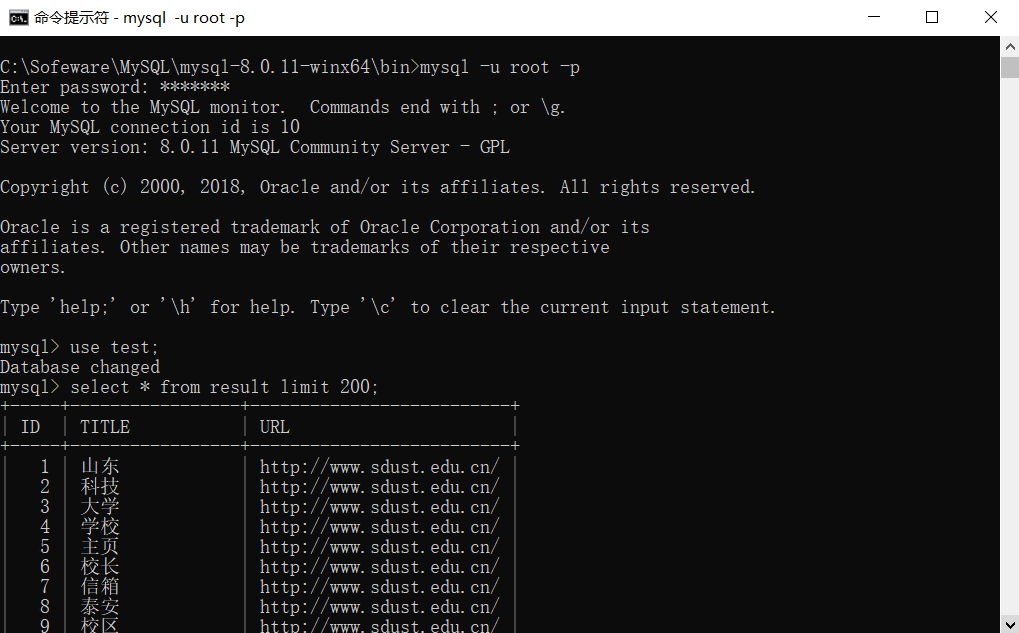
print(Rank(urls))

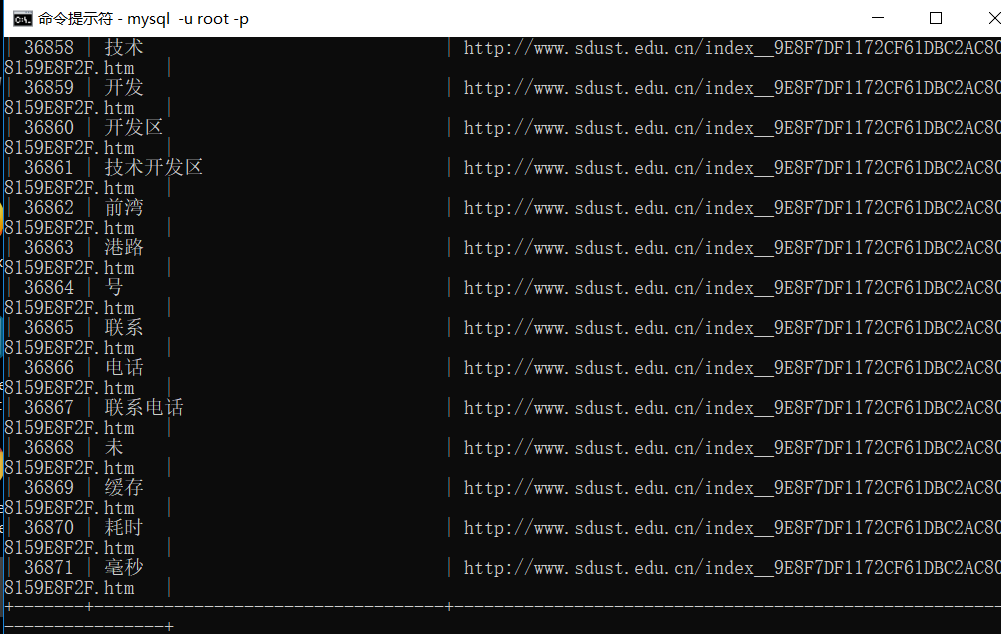
代码实际效果如下： 

**第四章 测试和演示**

1.MySQL数据库里的数据记录

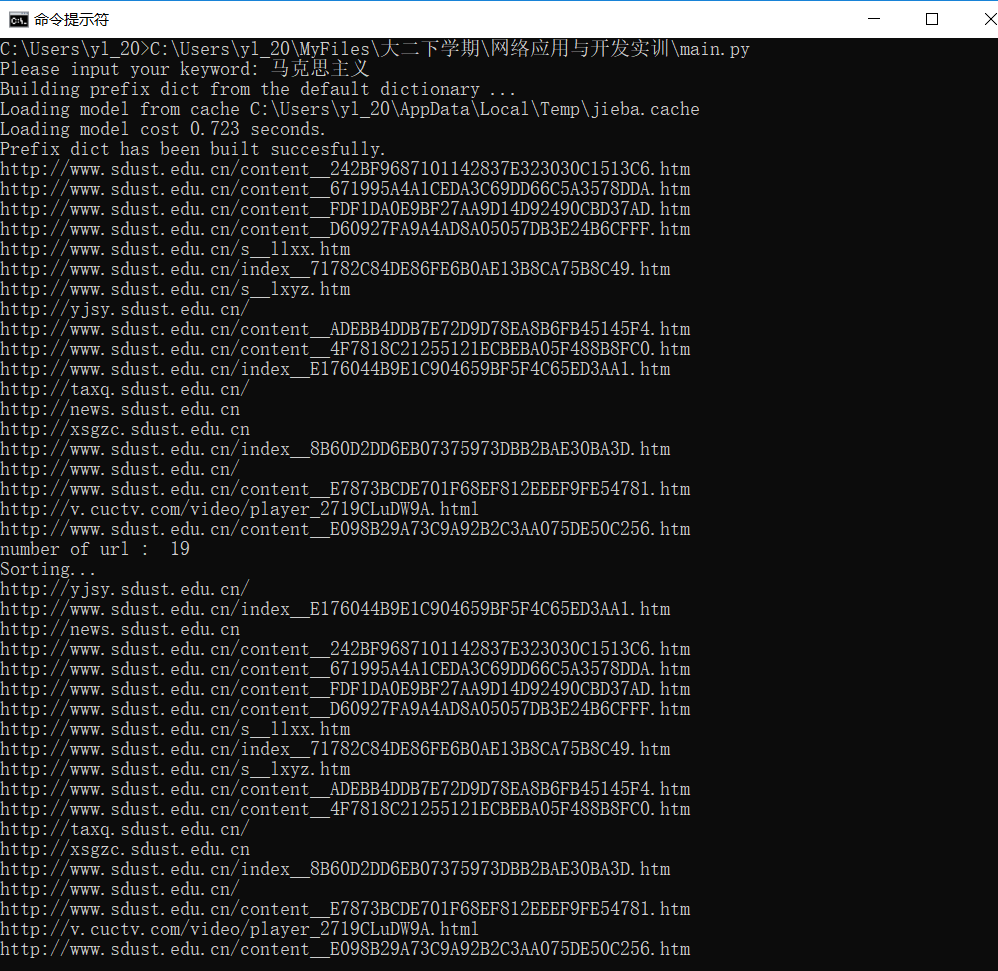
title为中文分词,URL为对应的url链接。





2.搜索“马克思主义”，结果如下：

上半部分为未排名的URL链接，下半部分为排名后的URL链接



**第五章 总结**

**1.工欲善其事必先利其器**

1. 想要爬取网站上的信息，必须有一个好的工具。firefox和chrome都有不错的工具，.

2. F12：快捷键，（更多工具——开发者工具）。可以直接查看当前页面的html结构。**有一点要注意**，当前页面的html结构可能是Js动态生成的。比如淘宝网页的Josn数据源，但在开发者工具下是HTML结构。这个时候使用鼠标右键——查看源代码，可以看到json数据。

3. console控制台，在这里可以看到一些与服务器的交互信息，上图中蓝色所指为清空，在此界面下，点击网页上的链接、按键或是F5刷新，可以看到与网页的交互信息。点击相应console下新出现的链接，可直接跳转到对应信息条目下

4. **requests + bs4 + re**——分别是网页下载、BeautifulSoup提取网页结构信息和正则表达式。这三个为对应python库名，网上相关内容很多。

**2.心得体会**

通过这个项目，我们更加深入学习了python爬虫的技术，阅读了很多关于python爬虫的书籍，python提供了大量的网络请求工具，我们可以通过对应的工具快速实现网络爬虫。最终超额的实现了预期目标，不仅仅局限于爬虫，还做了网页索引和简单搜索。同时我们也考虑了爬虫的优化，还有爬到的网页PageRank排名。我们也认识到爬虫的性能，拓展性，时效性是一个重要的判断标准，也引发了我们的思考

1. 每秒钟能够抓取的网页数量是衡量爬虫性能的重要指标。这也是爬虫开发者需要好好考虑的一个问题。如和在保证友好性的同时尽可能的提高自身的效率，还需集体问题具体分析。
2. 虫需要抓取的网页数最巨大，即使单个爬虫的性能很高，要将所有网页都下载到本地，仍然需要相当长的时间周期，为了能够尽可能缩短抓取周期，爬虫系统应该有很好的可扩展性，即很容易通过增加抓取服务器和爬虫数量来达到此目的
3. 虫是有很强的针对性的。爬虫下载网页我们可以很容易的抽象出统一的模板，对于网站的解析而言，却很难有统一的格式。解析操作是在我们下载的网页中提取我们需要的内容，网站的设计不同，提取的内容不同，相关的操作也存在着很大的差异。
4. 访问某个网站，首先我们要获取网站的连接。如果我们要抓取的数据我们无法访问到其中的内容，或者是无法从一个页面转到下一个页面（也就是下载规则不明确），做不到连续下载。