**بسمه تعالی**

**مستندات پروژه پایانی کارشناسی با موضوع طراحی و پیاده سازی موتور جستجوگر وب**

استاد راهنما: دکتر مهدی یعقوبی

دانشجو: مسلم آخوندی

دانشکده فنی مهندسی گرگان – دانشگاه گلستان

رشته مهندسی کامپیوتر – گرایش مهندسی نرم افزار (سیستم‌های اطلاعاتی)

**فهرست مطالب:**

|  |  |
| --- | --- |
| آشنایی با موتورهای جستجوگر | ۲ |
| توضیحات پروژه | ۳ |
| بخش خزنده وب (WebCrawler) | ۴ |
| بخش ایندکسر (Indexer) | ۱۲ |
| بخش رتبه بندی (Ranker) | 38 |
| بخش جستجو (Search) | 52 |
| پایگاه داده (Database) | 57 |

# **آشنایی با موتورهای جستجوگر وب**

موتور جستُجوی وب به انگلیسی Web search engine  ابزاریست که به منظور جُستجو در وب برای بدست‌آوردن اطلاعات درخواست شده، به کار می‌رود. نتایج یافته شده به‌طور معمول در صفحه‌ای با عنوان صفحهٔ نتایج جستجو(Search engine results page) فهرست می‌شوند.

با استفاده از کلمهٔ کلیدی(keyword) که در واقع توضیحی است کوتاه دربارهٔ آنچه لازم است در اینترنت پیدا شود، کلمه کلیدی باید تا آنجا که ممکن است کوتاه، جزیی و دقیق باشد. به غیر از وارد کردن کلمهٔ مستقیم، می‌توان با استفاده از عملگرهایی عمل جستجو را دقیق تر و منظم تر انجام داد.

**چگونگی عملکرد موتورهای جستجوی وب**

موتورهای جستجوی وب برای آنکه بتوانند به درخواست‌ها و جستجوهای کاربران پاسخ مناسبی بدهند باید دو کار مهم را انجام دهند:

* **خزیدن در وب‌سایت‌ها و ایندکس کردن صفحات**

موتورهای جستجو با استفاده از خزنده‌های خود می‌توانند تمامی صفحات و فایل‌های موجود در وب را ایندکس کنند. این خزنده‌ها با ورود به هر صفحه به ‌دنبال لینک‌ها هستند تا با دنبال کردن آن‌ها وارد صفحات جدید شوند. با این روش، موتورهای جستجو می‌توانند تمامی صفحات موجود در وب را ایندکس کنند.

* **رتبه‌بندی وب‌سایت‌ها و نمایش بهترین نتایج**

اصلی‌ترین وظیفهٔ موتورهای جستجوی وب، ارائهٔ بهترین و مرتبط‌ترین نتایج به کاربران است. موتورهای جستجو برای آنکه بتوانند بهترین نتایج را هنگام جستجوی یک عبارت خاص به کاربران نمایش دهند، قوانین و استانداردهایی برای وب‌سایت‌ها تعریف کرده‌اند تا بتوانند علاوه بر دسترسی ساده‌تر به صفحات وب‌سایت‌ها، محتوای موجود در صفحات را بهتر درک کنند. با این کار، موتورهای جستجو می‌توانند وب‌سایت‌هایی که دارای بیشترین ارتباط معنایی با عبارت جستجو شده توسط کاربر هستند در رتبه‌های بالاتری در نتایج جستجو قرار دهند.

موتورهای جستجو برای ایجاد نتیجه یک جستجو، تعدادی فعالیت انجام می‌دهند:

خزش یا Crawling فرایند واکشی تمام صفحات وب لینک شده به یک وب سایت؛ که با استفاده از نرم‌افزارهایی به نام خزشگر (Crawler)انجام می‌شود. خزشگر گوگل Google bot نام دارد.

اندیس گذاری یا Indexing فرایند ایجاد اندیس برای تمام صفحات واکشی شده و ذخیره و نگهداری این اندیس‌ها در یک پایگاه داده بسیار بزرگ که در آینده استفاده شوند. به عبارتی، فرایند اندیس گذاری از یک طرف شناسایی کلمات و عباراتی است که صفحه را به خوبی توصیف می‌کند و از طرف دیگر انتساب کلمات کلیدی یافت شده به صفحه می‌باشد.

پردازش یا Processing وقتی درخواست جستجو به موتور جستجو می‌رسد، موتور جستجو آن را پردازش می‌کند. در واقع موتور جستجو رشته جستجو را با صفحات اندیس شده در پایگاه داده مقایسه می‌کند.

محاسبه ارتباط یا Relevancy از آنجایی که ممکن است بیشتر از یک صفحه حاوی رشته جستجو باشند، بنابراین موتور جستجو شروع به محاسبه ارتباط هر صفحه با رشته جستجو می‌کند. برگرداندن نتیجه: آخرین گامی که موتور جستجو انجام می‌دهد، بازیابی بهترین نتیجه منطبق می‌باشد. به عبارت ساده، این فعالیت چیزی نیست جز نمایش اطلاعات در مرورگر.

از نرم افزارهای موتور جستجوگر وب می توان به Google, Bing, Yahoo Search, baidu, Yandex ,… اشاره کرد.

# **توضیحات پروژه**

موضوع پروژه: طراحی و پیاده سازی موتور جستجوگر وب

لینک گیت هاب: <https://github.com/ariantron/WebSearchEngine>

**بخش ها:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ردیف** | **عنوان** | **زبان برنامه نویسی** | **نرم افزار پایگاه داده** |
| ۱ | خزنده وب (WebCrawler) | Java | MySQL |
| ۲ | ایندکسر (Indexer) | Java | MySQL |
| ۳ | رتبه بندی (Ranking) | Java | MySQL |
| ۴ | جستجو (Search) | PHP (Laravel framework) | MySQL |

|  |  |  |  |
| --- | --- | --- | --- |
| **ردیف** | **عنوان** | **موضوع** | **لینک** |
| ۱ | Jsoup | پردازش صفحات html | <https://mvnrepository.com/artifact/org.jsoup/jsoup> |
| 2 | htmlcleaner | پردازش صفحات html | <https://mvnrepository.com/artifact/mysql/mysql-connector-java> |
| 3 | mysql-connector-java | ارتباط با دیتابیس MySQL در جاوا | <https://mvnrepository.com/artifact/net.sourceforge.htmlcleaner/htmlcleaner> |

**کتابخانه‌های مورد استفاده:**

**بخش خزنده وب (WebCrawler)**

در این بخش، نتیجتاً مجموعه ای از لینک‌ها ساخته می‌شود که محتوای html هر یک از آن‌ها در بخش ایندکسر تجزیه و تحلیل می گردد و اطلاعاتی که در بخش رتبه‌بندی مورد کاربرد است توسط ایندکسر، در پایگاه داده فهرست می‌شود.

در این بخش از دو کلاس Spider و SpiderLeg جهت انجام عملیات Crawling استفاده شده است. پارامتر ورودی متد سازنده کلاس SpiderLeg ، یک متغیر از نوع String است که نشان دهنده لینکی است که عملیات crawling در مورد آن می‌خواهد انجام شود. متد سازنده پس از بررسی ارتباط نرم‌افزار با شبکه اینترنت محتوای صفحه html مربوط به لینک را پردازش می‌کند و لینک‌های موجود در آن را که عمدتاً در تگ های a می‌باشند را می‌یابد. لینک‌های یافت‌شده در صورتی که قبلاً به پایگاه داده افزوده نشده باشند، یک رکورد مرتبط با آن‌ها در جدول urls در پایگاه داده ثبت می‌گردد. همچنین ارتباط بین لینک‌ها در جدول links در پایگاه داده ثبت می‌شود. این جدول مشخص می‌کند که هر یک از لینک‌ها برای اولین بار از طریق کدام یک از لینک‌ها یافت شده است. متدهای extractUrlsFromSite و getValidateUrls به ترتیب عملیات استخراج لینک‌ها از یک url و دریافت url های معتبر از یک صفحه html را در کلاس SpiderLegانجام می دهند.

در کلاس Spider، از کلاس SpiderLeg در متدهای start آبجکت تعریف می‌گردد. ورودی متدهای سازنده این آبجکتها توسط کلاس Spider فراهم می‌شود به این ترتیب که در کلاس Spider با به کار بردن متد nextUrl هر بار ورودی جدیدی با یافتن لینک‌های جدید فراهم می‌شود و به متد سازنده آبجکتهای کلاس SpiderLeg پاس داده می‌شود. در متد start در کلاس Spider این قابلیت وجود دارد که یک وب‌سایت تا تعداد مشخصی از لینک‌ها عملیات ‌crawling انجام شود.

وب سایت هایی که جهت Crawling انتخاب می‌شود می‌تواند لیستی از پیش تهیه شده باشد و یا با یک ‌dns server در ارتباط باشد و دریافت اطلاعات داشته باشد. در این پروژه به صورت آزمایشی وب سایت‌های دانشگاهی ایالات متحده آمریکا به عنوان منبع Crawling در نظر گرفته شده است.

در کلاس DataAccess ، توسط متد getConnection، یک آبجکت از کلاس Connection ایجاد می شود. این کلاس مسئولیت برقراری ارتباط نرم افزار جاوا با دیتابیس MySQL را بر عهده دارد. کوئری های موردنیاز جهت انجام متدهای مربوط به Crawling ابتدا در متغیرهای PreparedStatement ذخیره و سپس با متد excute اجرا می‌شوند.

کلاس utill شامل متدهای متفرقه ای است که در طول برنامه نویسی این بخش مورد کاربرد بوده است که شامل متدهای getPageTitleFromHtml، getPageTitleFromUrl، getUrlHtml می باشد که به ترتیب عملیات استخراج عنوان از محتوای html ، استخراج عنوان از url و استخراج محتوای html از url را بر عهده دارند.

در ادامه کدهای هر یک کلاس مذکور، آورده شده است.

**کلاس Spider :**

**package** Crawler;  
  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** Spider {  
  
 **private** List<String> **hasVisited** = **new** ArrayList<>();  
 **private** List<String> **toVisit** = **new** ArrayList<>();  
  
 **public static** Spider getInstance() {  
 **return new** Spider();  
 }  
  
 **public static void** start() **throws** SQLException, ClassNotFoundException {  
 DataAccess dataAccess=DataAccess.*getAccess*();  
 dataAccess.resetCrawlingTables();  
 ResultSet resultSet = dataAccess.getWebsites();  
 String url;  
 **while** (**true**) {  
 resultSet.first();  
 **do** {  
 url = resultSet.getString(**"url"**);  
 *getInstance*().start(url, 10);  
 } **while** (resultSet.next());  
 }  
 }  
  
 **public void** start(String url, **int** size) **throws** SQLException, ClassNotFoundException {  
 **toVisit**.add(url);  
  
 **while** (**hasVisited**.size() < size) {  
 String next = nextUrl();  
 **if** (next != **null**) {  
 SpiderLeg leg = **new** SpiderLeg(next);  
 **if** (**toVisit**.size() < 10000) **toVisit**.addAll(leg.getLinks());  
 } **else break**;  
 }  
*// System.out.println("[Done] Visited " + hasVisited.size() + " web page(s).");* utill.*writeALine*();  
 }  
  
 **public void** start(String url) **throws** SQLException, ClassNotFoundException {  
 **toVisit**.add(url);  
  
 **while** (**true**) {  
 String next = nextUrl();  
 **if** (next != **null**) {  
 SpiderLeg leg = **new** SpiderLeg(next);  
 **if** (**toVisit**.size() < 10000) **toVisit**.addAll(leg.getLinks());  
 } **else break**;  
 }  
*// System.out.println("[Done] Visited " + hasVisited.size() + " web page(s).");* utill.*writeALine*();  
 }  
  
 **private** String nextUrl() {  
 String nextUrl;  
 **do** {  
 **if** (**toVisit**.size() > 0) nextUrl = **toVisit**.remove(0);  
 **else return null**;  
 } **while** (**hasVisited**.contains(nextUrl));  
 **hasVisited**.add(nextUrl);  
 **return** nextUrl;  
 }  
}

**کلاس SpiderLeg :**

**package** Crawler;  
  
**import** org.jsoup.Connection;  
**import** org.jsoup.Jsoup;  
**import** org.jsoup.nodes.Document;  
**import** org.jsoup.nodes.Element;  
**import** org.jsoup.select.Elements;  
  
**import** java.io.IOException;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** SpiderLeg {  
  
 **private static final** String ***USER\_AGENT*** = **"Mozilla/5.0 (Windows NT 6.1; WOW64)"** +  
 **" AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.112 Safari/535.1"**;  
 **private static int** *count*;  
 **private** List<String> **links** = **new** ArrayList<>();  
 **private** DataAccess **dataAccess**;  
  
 **public** SpiderLeg(String url) **throws** SQLException, ClassNotFoundException {  
 **dataAccess** = DataAccess.*getAccess*();  
 *count*++;  
 System.***out***.println(**"[Number] "** + *count*);  
 **boolean** success;  
 **try** {  
 Connection connection = Jsoup.*connect*(url).userAgent(***USER\_AGENT***);  
 Document htmlDocument = connection.get();  
 **if** (connection.response().statusCode() == 200) { *// 200 is the HTTP OK status code* **if** (connection.response().contentType().contains(**"text/html"**)) {  
 String title = utill.*getPageTitleFromUrl*(url);  
 **if** (!isUrlAlreadyAdded(url)) **dataAccess**.addNode(url, title);  
 **dataAccess**.setIndexTrue(url);  
 System.***out***.println(**"[Visit] "** + url);  
 ArrayList<String> linksOnPage = extractUrlsFromSite(htmlDocument.baseUri());  
 System.***out***.println(**" - Found ("** + linksOnPage.size() + **") links"**);  
 System.***out***.println(**"title: "** + title);  
 utill.*writeALine*();  
 **long** sourceId = getID(url);  
 **for** (String link : linksOnPage) {  
 **links**.add(link);  
 **if** (!isUrlAlreadyAdded(link)) **dataAccess**.addNode(link, utill.*getPageTitleFromUrl*(link));  
 **long** targetId = getID(link);  
 **if** (sourceId != targetId) **dataAccess**.createLink(sourceId, targetId);  
 }  
 success = **true**;  
 } **else** {  
 System.***out***.println(**"[FailureA] "** + url);  
 utill.*writeALine*();  
 success = **false**;  
 }  
 }  
 } **catch** (Exception e) {  
 System.***out***.println(**"[FailureB] "** + url);  
 utill.*writeALine*();  
 success = **false**;  
 }  
 }  
  
 **private** ArrayList<String> extractUrlsFromSite(String url) **throws** IOException {  
 Document doc = Jsoup.*connect*(url).userAgent(***USER\_AGENT***).get();  
 Elements links = doc.getElementsByTag(**"a"**);  
 **return** getValidateUrls(doc, links);  
 }  
  
 **private** ArrayList<String> getValidateUrls(Document doc, Elements links) {  
 ArrayList<String> urls = **new** ArrayList<>();  
  
 **for** (Element link : links) {  
 String linkUrl = link.attr(**"href"**);  
  
 **if** (linkUrl.length() > 0) {  
 **if** (linkUrl.length() < 4) {  
 linkUrl = doc.baseUri() + linkUrl.substring(1);  
 } **else if** (!linkUrl.substring(0, 4).equals(**"http"**)) {  
 linkUrl = doc.baseUri() + linkUrl.substring(1);  
 }  
 }  
 **if** (linkUrl.isEmpty()) {  
 **continue**;  
 }  
 urls.add(linkUrl);  
 }  
  
 **return** urls;  
 }  
  
 **private boolean** isUrlAlreadyAdded(String url) **throws** SQLException {  
 **return dataAccess**.isNodeAlreadyExisting(url);  
 }  
  
 **private long** getID(String url) **throws** SQLException {  
 **return dataAccess**.getID(url);  
 }  
  
 **public** List<String> getLinks() {  
 **return links**;  
 }  
  
}

**کلاس DataAccess :**

**package** Crawler;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
  
**public class** DataAccess {  
 **private static** DataAccess *access*;  
  
 **private** PreparedStatement **addNewNodeStatement**;  
 **private** PreparedStatement **findIdForUrlStatement**;  
 **private** PreparedStatement **setIndexTrueStatement**;  
 **private** PreparedStatement **createLinkStatement**;  
 **private** PreparedStatement[] **resetCrawlingTablesStatements**;  
 **private** PreparedStatement **getWebsites**;  
  
 **private** DataAccess() **throws** SQLException {  
 init();  
 }  
  
 **public static** Connection getConnection() {  
 Connection connection = **null**;  
 **try** {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 connection = DriverManager.*getConnection*(  
 **"jdbc:mysql://localhost:3306/search\_engine"**, **"root"**, **""**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** connection;  
 }  
  
 **public static** DataAccess getAccess() **throws** SQLException, ClassNotFoundException {  
 **if** (*access* == **null**) {  
 *access* = **new** DataAccess();  
 }  
 **return** *access*;  
 }  
  
 **private void** init() **throws** SQLException {  
 Connection dbConnection = *getConnection*();  
 **findIdForUrlStatement** = dbConnection.prepareStatement(**"SELECT id FROM urls WHERE url=?;"**);  
 **addNewNodeStatement** = dbConnection.prepareStatement(**"INSERT INTO urls (url,title) VALUES(?,?);"**);  
 **createLinkStatement** = dbConnection.prepareStatement(**"INSERT INTO links (source, target) VALUES(?,?);"**);  
 **resetCrawlingTablesStatements** = **new** PreparedStatement[]{  
 dbConnection.prepareStatement(**"DELETE FROM urls;"**),  
 dbConnection.prepareStatement(**"DELETE FROM links;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE urls AUTO\_INCREMENT = 1;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE links AUTO\_INCREMENT = 1;"**),  
 };  
 **setIndexTrueStatement** = dbConnection.prepareStatement(**"UPDATE urls SET is\_index=TRUE WHERE url=?"**);  
 **getWebsites** = dbConnection.prepareStatement(**"SELECT url FROM websites"**);  
 }  
  
 **public boolean** addNode(String url,String title) **throws** SQLException {  
 **addNewNodeStatement**.setString(1, url);  
 **addNewNodeStatement**.setString(2, title);  
 **return addNewNodeStatement**.execute();  
 }  
  
 **public boolean** isNodeAlreadyExisting(String url) **throws** SQLException {  
 **findIdForUrlStatement**.setString(1, url);  
 **return findIdForUrlStatement**.executeQuery().next();  
 }  
  
 **public long** getID(String url) **throws** SQLException {  
 **findIdForUrlStatement**.setString(1, url);  
 ResultSet resultSet = **findIdForUrlStatement**.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"id"**);  
 }  
  
 **public boolean** setIndexTrue(String url) **throws** SQLException {  
 **setIndexTrueStatement**.setString(1, url);  
 **return setIndexTrueStatement**.executeUpdate() > 0;  
 }  
  
 **public boolean** createLink(**long** sourceID, **long** targetID) **throws** SQLException {  
 **createLinkStatement**.setLong(1, sourceID);  
 **createLinkStatement**.setLong(2, targetID);  
 **return createLinkStatement**.execute();  
 }  
  
 **public void** resetCrawlingTables() **throws** SQLException {  
 executeMultiStatement(**resetCrawlingTablesStatements**);  
 }  
  
 **private void** executeMultiStatement(PreparedStatement[] preparedStatements) **throws** SQLException {  
 **for** (PreparedStatement preparedStatement : preparedStatements) {  
 preparedStatement.executeUpdate();  
 }  
 }  
  
 **public** ResultSet getWebsites() **throws** SQLException {  
 **return getWebsites**.executeQuery();  
 }  
}

**کلاس utill :**

**package** Crawler;  
  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.net.URL;  
**import** java.net.URLConnection;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** utill {  
 **public static void** writeALine() {  
 System.***out***.println(**"--------------------"**);  
 }  
  
 **public static** String getPageTitleFromHtml(String html) {  
 Pattern p = Pattern.*compile*(**"<head>.\*?<title>(.\*?)</title>.\*?</head>"**, Pattern.***DOTALL***);  
 Matcher m = p.matcher(html);  
 String title = **""**;  
 **while** (m.find()) {  
 title = m.group(1);  
 }  
 **return** title;  
 }  
  
 **public static** String getPageTitleFromUrl(String url) **throws** IOException {  
 **return** *getPageTitleFromHtml*(*getUrlHtml*(url));  
 }  
  
 **public static** String getUrlHtml(String url) **throws** IOException {  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **try** {  
 URL theUrl = **new** URL(url);  
 URLConnection conn = theUrl.openConnection();  
 BufferedReader br = **new** BufferedReader(  
 **new** InputStreamReader(conn.getInputStream()));  
 String inputLine;  
 **while** ((inputLine = br.readLine()) != **null**) {  
 stringBuilder.append(inputLine);  
 }  
 br.close();  
 } **catch** (Exception ignored) {  
 }  
 **return** stringBuilder.toString();  
 }  
}

**بخش ایندکسر:**

در این بخش از پروژه، اطلاعات موردنیاز جهت رتبه بندی لینک های یافت شده در بخش Crawler، جمع‌آوری و فهرست می‌شود. کلاس اصلی مورد کاربرد در این بخش، کلاس Indexer می‌باشد. این کلاس شامل متدهایی است که مجموعه کلمات محتوانی متنی فایل های html را استخراج می کند. کلمات بی اهمیت (stop words) مانند کلمات ربطی را حذف می‌کنند و در دیتابیس با ذکر تعدادشان در هر url آن را ذخیره می کنند.

متد getKeywordsFromUrl : با دریافت آدرس url مجموعه ی کلمات کلیدی محتوای متنی فایل html مربوط به url به دست می‌آید.

متد insertWords : فرایند ذخیره کلمه در دیتابیس را انجام می‌دهد. پس از اینکه کنترل شد که کلمه قبلاً در دیتابیس اضافه نشده است. آن کلمه به عنوان یک رکورد در جدول words ثبت می‌گردد. همچنین پس از هر بار تکرار کلمه در url به فیلد freq آن کلمه در جدول word\_doc یک واحد نیز اضافه می‌گردد.

متد websitesIndex : مجموعه url های معتبر (در زمان دسترسی به محتوا، کاربر با خطایی مثل ۴۰۴ مواجه نشود.) را از دیتابیس دریافت و با بهره گیری از متدهای فوق عملیات indexing را در مورد هر یک از url ها انجام می‌دهد.

متد start: با فراخوانی این متد در یک حلقه ی بینهایت (از این رو که محتوای url ها دائماً در حال تغییر هستند.) عملیات indexing لینک های دیتابیس انجام می‌شود.

کلاس EnglishStopWords: مجموعه لغات کم اهمیت زبان انگلیسی را در یک آرایه استاتیک ذخیره کرده است. و امکان دسترسی به این آرایه از طریق این کلاس داریم.

کلاس Stemmer: این کلاس ریشه کلمات را با به کارگیری قواعدی در دستور زبان می‌یابد.

کلاس Tokenizer: این کلاس با استفاده از کلاس Stemmer علاوه بر ریشه یابی کلمات، کلمات stop words و خالی را حذف می کند و برای عملیات indexing آماده می‌کند.

کلاس utill: شامل سه متد متفرقه writeALine، textToWordsList و getUrlHtml است که به ترتیب رسم یک خط ساده، تبدیل متن به لیست کلمات و استخراج محتوای html از یک url را انجام می‌‌دهند.

کلاس DataAccess : در این کلاس توسط متد getConnection، یک آبجکت از کلاس Connection ایجاد می شود. این کلاس مسئولیت برقراری ارتباط نرم افزار جاوا با دیتابیس MySQL را بر عهده دارد. کوئری های موردنیاز جهت انجام متدهای مربوط به Indexing ابتدا در متغیرهای PreparedStatement ذخیره و سپس با متد excute اجرا می‌شوند.

**کلاس Indexer :**

**package** Indexer;  
  
**import** org.jsoup.Jsoup;  
**import** org.jsoup.safety.Whitelist;  
  
**import** java.io.IOException;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
  
**public class** Indexer {  
 **private** DataAccess **dataAccess**;  
  
 **private** Indexer() **throws** SQLException {  
 **this**.**dataAccess** = DataAccess.*getAccess*();  
 }  
  
 **public static** Indexer getInstance() **throws** SQLException {  
 **return new** Indexer();  
 }  
  
 **private static** String[] getKeywordsFromUrl(String url) {  
 String html = utill.*getUrlHtml*(url);  
 String text = Jsoup.*clean*(html, Whitelist.*simpleText*());  
 ArrayList<String> arrayList = utill.*textToWordsList*(text);  
 Tokenizer tokenizer = **new** Tokenizer(arrayList, EnglishStopWords.*stopWords*);  
 **return** tokenizer.getProcessedWords().split(**" "**);  
 }  
  
 **public static void** start() **throws** SQLException {  
 DataAccess.*getAccess*().resetIndexingTables();  
 **while** (**true**)  
 *getInstance*().websitesIndex();  
 }  
  
 **private void** insertWords(String url) **throws** SQLException {  
 String[] words = *getKeywordsFromUrl*(url);  
 **long** docUrlId = **dataAccess**.getID(url);  
 **int** docSize = words.**length**;  
 **if** (!**dataAccess**.isAlreadyInsertedInDocSize(docUrlId))  
 **dataAccess**.insertDocSize(docUrlId, docSize);  
 **boolean** isAlreadyAdded;  
 **long** wordId;  
 **for** (String item : words) {  
 isAlreadyAdded = **dataAccess**.isWordAlreadyExisting(item);  
 **if** (isAlreadyAdded) {  
 wordId = **dataAccess**.getWordId(item);  
 **if** (**dataAccess**.isAlreadyWordAddedToUrl(wordId, docUrlId)) {  
 **dataAccess**.increaseWordFreq(docUrlId, wordId);  
 System.***out***.println(**"update word--doc:"** + docUrlId + **",word:"** + item);  
 } **else** {  
 **dataAccess**.addWord2(wordId, docUrlId, 1);  
 System.***out***.println(**"new word--doc:"** + docUrlId + **",word:"** + item);  
 }  
 } **else** {  
 **dataAccess**.addWord(item);  
 wordId = **dataAccess**.getWordId(item);  
 **dataAccess**.addWord2(wordId, docUrlId, 1);  
 System.***out***.println(**"new word--doc:"** + docUrlId + **",word:"** + item);  
 }  
 }  
 }  
  
 **private void** websitesIndex() **throws** SQLException {  
 ResultSet resultSet = **dataAccess**.getRightUrls();  
 String url;  
 **int** id;  
 resultSet.first();  
 **do** {  
 url = resultSet.getString(**"url"**);  
 id = resultSet.getInt(**"id"**);  
 insertWords(url);  
 System.***out***.println(**"url -> id:"** + id + **"\n"** + url);  
 utill.*writeALine*();  
 } **while** (resultSet.next());  
 }  
  
}

**کلاس Stemmer:**

**package** Indexer;*/\*  
  
 Porter stemmer in Java. The original paper is in  
  
 Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,  
 no. 3, pp 130-137,  
  
 See also http://www.tartarus.org/~martin/PorterStemmer  
  
 History:  
  
 Release 1  
  
 Bug 1 (reported by Gonzalo Parra 16/10/99) fixed as marked below.  
 The words 'aed', 'eed', 'oed' leave k at 'a' for step 3, and b[k-1]  
 is then out outside the bounds of b.  
  
 Release 2  
  
 Similarly,  
  
 Bug 2 (reported by Steve Dyrdahl 22/2/00) fixed as marked below.  
 'ion' by itself leaves j = -1 in the test for 'ion' in step 5, and  
 b[j] is then outside the bounds of b.  
  
 Release 3  
  
 Considerably revised 4/9/00 in the light of many helpful suggestions  
 from Brian Goetz of Quiotix Corporation (brian@quiotix.com).  
  
 Release 4  
  
\*/  
  
//import java.io.\*;  
  
/\*\*  
 \* Stemmer, implementing the Porter Stemming Algorithm  
 \* <p>  
 \* The Stemmer class transforms a word into its root form. The input  
 \* word can be provided a character at time (by calling add()), or at once  
 \* by calling one of the various stem(something) methods.  
 \*/***class** Stemmer {  
 **private static final int *INC*** = 50;  
 **private char**[] **b**;  
 **private int i**, */\* offset into b \*/* **i\_end**, */\* offset to end of stemmed word \*/* **j**, **k**;  
  
 */\* unit of size whereby b is increased \*/* **public** Stemmer() {  
 **b** = **new char**[***INC***];  
 **i** = 0;  
 **i\_end** = 0;  
 }  
  
 */\*\*  
 \* Add a character to the word being stemmed. When you are finished  
 \* adding characters, you can call stem(void) to stem the word.  
 \*/* **public void** add(**char** ch) {  
 **if** (**i** == **b**.**length**) {  
 **char**[] new\_b = **new char**[**i** + ***INC***];  
 **for** (**int** c = 0; c < **i**; c++) new\_b[c] = **b**[c];  
 **b** = new\_b;  
 }  
 **b**[**i**++] = ch;  
 }  
  
  
 */\*\*  
 \* Adds wLen characters to the word being stemmed contained in a portion  
 \* of a char[] array. This is like repeated calls of add(char ch), but  
 \* faster.  
 \*/* **public void** add(**char**[] w, **int** wLen) {  
 **if** (**i** + wLen >= **b**.**length**) {  
 **char**[] new\_b = **new char**[**i** + wLen + ***INC***];  
 **for** (**int** c = 0; c < **i**; c++) new\_b[c] = **b**[c];  
 **b** = new\_b;  
 }  
 **for** (**int** c = 0; c < wLen; c++) **b**[**i**++] = w[c];  
 }  
  
 */\*\*  
 \* After a word has been stemmed, it can be retrieved by toString(),  
 \* or a reference to the internal buffer can be retrieved by getResultBuffer  
 \* and getResultLength (which is generally more efficient.)  
 \*/* **public** String toString() {  
 **return new** String(**b**, 0, **i\_end**);  
 }  
  
 */\*\*  
 \* Returns the length of the word resulting from the stemming process.  
 \*/* **public int** getResultLength() {  
 **return i\_end**;  
 }  
  
 */\*\*  
 \* Returns a reference to a character buffer containing the results of  
 \* the stemming process. You also need to consult getResultLength()  
 \* to determine the length of the result.  
 \*/* **public char**[] getResultBuffer() {  
 **return b**;  
 }  
  
 */\* cons(i) is true <=> b[i] is a consonant. \*/* **private final boolean** cons(**int** i) {  
 **switch** (**b**[i]) {  
 **case 'a'**:  
 **case 'e'**:  
 **case 'i'**:  
 **case 'o'**:  
 **case 'u'**:  
 **return false**;  
 **case 'y'**:  
 **return** (i == 0) ? **true** : !cons(i - 1);  
 **default**:  
 **return true**;  
 }  
 }  
  
 */\* m() measures the number of consonant sequences between 0 and j. if c is  
 a consonant sequence and v a vowel sequence, and <..> indicates arbitrary  
 presence,  
  
 <c><v> gives 0  
 <c>vc<v> gives 1  
 <c>vcvc<v> gives 2  
 <c>vcvcvc<v> gives 3  
 ....  
 \*/* **private final int** m() {  
 **int** n = 0;  
 **int** i = 0;  
 **while** (**true**) {  
 **if** (i > **j**) **return** n;  
 **if** (!cons(i)) **break**;  
 i++;  
 }  
 i++;  
 **while** (**true**) {  
 **while** (**true**) {  
 **if** (i > **j**) **return** n;  
 **if** (cons(i)) **break**;  
 i++;  
 }  
 i++;  
 n++;  
 **while** (**true**) {  
 **if** (i > **j**) **return** n;  
 **if** (!cons(i)) **break**;  
 i++;  
 }  
 i++;  
 }  
 }  
  
 */\* vowelinstem() is true <=> 0,...j contains a vowel \*/* **private final boolean** vowelinstem() {  
 **int** i;  
 **for** (i = 0; i <= **j**; i++) **if** (!cons(i)) **return true**;  
 **return false**;  
 }  
  
 */\* doublec(j) is true <=> j,(j-1) contain a double consonant. \*/* **private final boolean** doublec(**int** j) {  
 **if** (j < 1) **return false**;  
 **if** (**b**[j] != **b**[j - 1]) **return false**;  
 **return** cons(j);  
 }  
  
 */\* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant  
 and also if the second c is not w,x or y. this is used when trying to  
 restore an e at the end of a short word. e.g.  
  
 cav(e), lov(e), hop(e), crim(e), but  
 snow, box, tray.  
  
 \*/* **private final boolean** cvc(**int** i) {  
 **if** (i < 2 || !cons(i) || cons(i - 1) || !cons(i - 2)) **return false**;  
 {  
 **int** ch = **b**[i];  
 **if** (ch == **'w'** || ch == **'x'** || ch == **'y'**) **return false**;  
 }  
 **return true**;  
 }  
  
 **private final boolean** ends(String s) {  
 **int** l = s.length();  
 **int** o = **k** - l + 1;  
 **if** (o < 0) **return false**;  
 **for** (**int** i = 0; i < l; i++) **if** (**b**[o + i] != s.charAt(i)) **return false**;  
 **j** = **k** - l;  
 **return true**;  
 }  
  
 */\* setto(s) sets (j+1),...k to the characters in the string s, readjusting  
 k. \*/* **private final void** setto(String s) {  
 **int** l = s.length();  
 **int** o = **j** + 1;  
 **for** (**int** i = 0; i < l; i++) **b**[o + i] = s.charAt(i);  
 **k** = **j** + l;  
 }  
  
 */\* r(s) is used further down. \*/* **private final void** r(String s) {  
 **if** (m() > 0) setto(s);  
 }  
  
 */\* step1() gets rid of plurals and -ed or -ing. e.g.  
  
 caresses -> caress  
 ponies -> poni  
 ties -> ti  
 caress -> caress  
 cats -> cat  
  
 feed -> feed  
 agreed -> agree  
 disabled -> disable  
  
 matting -> mat  
 mating -> mate  
 meeting -> meet  
 milling -> mill  
 messing -> mess  
  
 meetings -> meet  
  
 \*/* **private final void** step1() {  
 **if** (**b**[**k**] == **'s'**) {  
 **if** (ends(**"sses"**)) **k** -= 2;  
 **else if** (ends(**"ies"**)) setto(**"i"**);  
 **else if** (**b**[**k** - 1] != **'s'**) **k**--;  
 }  
 **if** (ends(**"eed"**)) {  
 **if** (m() > 0) **k**--;  
 } **else if** ((ends(**"ed"**) || ends(**"ing"**)) && vowelinstem()) {  
 **k** = **j**;  
 **if** (ends(**"at"**)) setto(**"ate"**);  
 **else if** (ends(**"bl"**)) setto(**"ble"**);  
 **else if** (ends(**"iz"**)) setto(**"ize"**);  
 **else if** (doublec(**k**)) {  
 **k**--;  
 {  
 **int** ch = **b**[**k**];  
 **if** (ch == **'l'** || ch == **'s'** || ch == **'z'**) **k**++;  
 }  
 } **else if** (m() == 1 && cvc(**k**)) setto(**"e"**);  
 }  
 }  
  
 */\* step2() turns terminal y to i when there is another vowel in the stem. \*/* **private final void** step2() {  
 **if** (ends(**"y"**) && vowelinstem()) **b**[**k**] = **'i'**;  
 }  
  
 */\* step3() maps double suffices to single ones. so -ization ( = -ize plus  
 -ation) maps to -ize etc. note that the string before the suffix must give  
 m() > 0. \*/* **private final void** step3() {  
 **if** (**k** == 0) **return**; */\* For Bug 1 \*/* **switch** (**b**[**k** - 1]) {  
 **case 'a'**:  
 **if** (ends(**"ational"**)) {  
 r(**"ate"**);  
 **break**;  
 }  
 **if** (ends(**"tional"**)) {  
 r(**"tion"**);  
 **break**;  
 }  
 **break**;  
 **case 'c'**:  
 **if** (ends(**"enci"**)) {  
 r(**"ence"**);  
 **break**;  
 }  
 **if** (ends(**"anci"**)) {  
 r(**"ance"**);  
 **break**;  
 }  
 **break**;  
 **case 'e'**:  
 **if** (ends(**"izer"**)) {  
 r(**"ize"**);  
 **break**;  
 }  
 **break**;  
 **case 'l'**:  
 **if** (ends(**"bli"**)) {  
 r(**"ble"**);  
 **break**;  
 }  
 **if** (ends(**"alli"**)) {  
 r(**"al"**);  
 **break**;  
 }  
 **if** (ends(**"entli"**)) {  
 r(**"ent"**);  
 **break**;  
 }  
 **if** (ends(**"eli"**)) {  
 r(**"e"**);  
 **break**;  
 }  
 **if** (ends(**"ousli"**)) {  
 r(**"ous"**);  
 **break**;  
 }  
 **break**;  
 **case 'o'**:  
 **if** (ends(**"ization"**)) {  
 r(**"ize"**);  
 **break**;  
 }  
 **if** (ends(**"ation"**)) {  
 r(**"ate"**);  
 **break**;  
 }  
 **if** (ends(**"ator"**)) {  
 r(**"ate"**);  
 **break**;  
 }  
 **break**;  
 **case 's'**:  
 **if** (ends(**"alism"**)) {  
 r(**"al"**);  
 **break**;  
 }  
 **if** (ends(**"iveness"**)) {  
 r(**"ive"**);  
 **break**;  
 }  
 **if** (ends(**"fulness"**)) {  
 r(**"ful"**);  
 **break**;  
 }  
 **if** (ends(**"ousness"**)) {  
 r(**"ous"**);  
 **break**;  
 }  
 **break**;  
 **case 't'**:  
 **if** (ends(**"aliti"**)) {  
 r(**"al"**);  
 **break**;  
 }  
 **if** (ends(**"iviti"**)) {  
 r(**"ive"**);  
 **break**;  
 }  
 **if** (ends(**"biliti"**)) {  
 r(**"ble"**);  
 **break**;  
 }  
 **break**;  
 **case 'g'**:  
 **if** (ends(**"logi"**)) {  
 r(**"log"**);  
 **break**;  
 }  
 }  
 }  
  
 */\* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. \*/* **private final void** step4() {  
 **switch** (**b**[**k**]) {  
 **case 'e'**:  
 **if** (ends(**"icate"**)) {  
 r(**"ic"**);  
 **break**;  
 }  
 **if** (ends(**"ative"**)) {  
 r(**""**);  
 **break**;  
 }  
 **if** (ends(**"alize"**)) {  
 r(**"al"**);  
 **break**;  
 }  
 **break**;  
 **case 'i'**:  
 **if** (ends(**"iciti"**)) {  
 r(**"ic"**);  
 **break**;  
 }  
 **break**;  
 **case 'l'**:  
 **if** (ends(**"ical"**)) {  
 r(**"ic"**);  
 **break**;  
 }  
 **if** (ends(**"ful"**)) {  
 r(**""**);  
 **break**;  
 }  
 **break**;  
 **case 's'**:  
 **if** (ends(**"ness"**)) {  
 r(**""**);  
 **break**;  
 }  
 **break**;  
 }  
 }  
  
 */\* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. \*/* **private final void** step5() {  
 **if** (**k** == 0) **return**; */\* for Bug 1 \*/* **switch** (**b**[**k** - 1]) {  
 **case 'a'**:  
 **if** (ends(**"al"**)) **break**;  
 **return**;  
 **case 'c'**:  
 **if** (ends(**"ance"**)) **break**;  
 **if** (ends(**"ence"**)) **break**;  
 **return**;  
 **case 'e'**:  
 **if** (ends(**"er"**)) **break**;  
 **return**;  
 **case 'i'**:  
 **if** (ends(**"ic"**)) **break**;  
 **return**;  
 **case 'l'**:  
 **if** (ends(**"able"**)) **break**;  
 **if** (ends(**"ible"**)) **break**;  
 **return**;  
 **case 'n'**:  
 **if** (ends(**"ant"**)) **break**;  
 **if** (ends(**"ement"**)) **break**;  
 **if** (ends(**"ment"**)) **break**;  
 */\* element etc. not stripped before the m \*/* **if** (ends(**"ent"**)) **break**;  
 **return**;  
 **case 'o'**:  
 **if** (ends(**"ion"**) && **j** >= 0 && (**b**[**j**] == **'s'** || **b**[**j**] == **'t'**)) **break**;  
 */\* j >= 0 fixes Bug 2 \*/* **if** (ends(**"ou"**)) **break**;  
 **return**;  
 */\* takes care of -ous \*/* **case 's'**:  
 **if** (ends(**"ism"**)) **break**;  
 **return**;  
 **case 't'**:  
 **if** (ends(**"ate"**)) **break**;  
 **if** (ends(**"iti"**)) **break**;  
 **return**;  
 **case 'u'**:  
 **if** (ends(**"ous"**)) **break**;  
 **return**;  
 **case 'v'**:  
 **if** (ends(**"ive"**)) **break**;  
 **return**;  
 **case 'z'**:  
 **if** (ends(**"ize"**)) **break**;  
 **return**;  
 **default**:  
 **return**;  
 }  
 **if** (m() > 1) **k** = **j**;  
 }  
  
 */\* step6() removes a final -e if m() > 1. \*/* **private final void** step6() {  
 **j** = **k**;  
 **if** (**b**[**k**] == **'e'**) {  
 **int** a = m();  
 **if** (a > 1 || a == 1 && !cvc(**k** - 1)) **k**--;  
 }  
 **if** (**b**[**k**] == **'l'** && doublec(**k**) && m() > 1) **k**--;  
 }  
  
 */\*\*  
 \* Stem the word placed into the Stemmer buffer through calls to add().  
 \* Returns true if the stemming process resulted in a word different  
 \* from the input. You can retrieve the result with  
 \* getResultLength()/getResultBuffer() or toString().  
 \*/* **public void** stem() {  
 **k** = **i** - 1;  
 **if** (**k** > 1) {  
 step1();  
 step2();  
 step3();  
 step4();  
 step5();  
 step6();  
 }  
 **i\_end** = **k** + 1;  
 **i** = 0;  
 }  
}

**کلاس Tokenizer:**

**package** Indexer;  
  
**import** java.util.ArrayList;  
  
*/\*\*  
 \** ***@author*** *brandonskane This class tokenizes, removes stop words and then uses  
 \* the Porter Stemming Algorithm (via the Stemmer class) to stem words.  
 \*/***public class** Tokenizer {  
  
 ArrayList<String> **allWords**;  
 ArrayList<String> **stopWords**;  
 ArrayList<String> **stemmedWords**;  
  
 */\*\*  
 \** ***@param seperatedWords*** *\* the words from the input file. They are potentially dirty but  
 \* will be cleaned  
 \** ***@param stopWords*** *\* the stop words list. They must be properly formatted prior  
 \*/* **public** Tokenizer(String[] seperatedWords, String[] stopWords) {  
 **this**.**allWords** = **new** ArrayList<String>();  
 **this**.**stopWords** = **new** ArrayList<String>();  
 **this**.**stemmedWords** = **new** ArrayList<String>();  
  
 tokenizer(seperatedWords);  
 removeEmptyWords();  
 System.***out***.println(**"Token"**);  
 System.***out***.println(**allWords**);  
 removeStopWords(stopWords);  
 System.***out***.println(**"Remove Stop"**);  
 System.***out***.println(**allWords**);  
 stemWords();  
 }  
  
 **public** Tokenizer(ArrayList<String> seperatedWords, String[] stopWords) {  
 **this**.**allWords** = **new** ArrayList<String>();  
 **this**.**stopWords** = **new** ArrayList<String>();  
 **this**.**stemmedWords** = **new** ArrayList<String>();  
  
 tokenizer(seperatedWords);  
 removeEmptyWords();  
*// System.out.println("Token");  
// System.out.println(allWords);* removeStopWords(stopWords);  
*// System.out.println("Remove Stop");  
// System.out.println(allWords);* stemWords();  
 }  
  
 */\*\*  
 \** ***@return*** *All of the final tokens from the stemmedWords list (the completed  
 \* product)  
 \*/* **public** String getProcessedWords() {  
 StringBuilder stringBuilder = **new** StringBuilder();  
  
 **for** (String word : **stemmedWords**) {  
 stringBuilder.append(word + **" "**);  
 }  
  
 **return** stringBuilder.toString();  
 }  
  
 */\*\*  
 \* The Stemmer class requires words to be build via chars. Go through each  
 \* word in the the allWords array and create Stemmer objects Then convert  
 \* back to a String and add to stemmedWords (final product state)  
 \*/* **private void** stemWords() {  
  
 **char**[] wordCharArray;  
 Stemmer s = **new** Stemmer();  
  
 **for** (String word : **allWords**) {  
 wordCharArray = word.toCharArray();  
  
 **for** (**int** i = 0; i < word.length(); i++) {  
 s.add(wordCharArray[i]);  
 }  
 s.stem(); *// call to "stem" the word* **stemmedWords**.add(s.toString());  
 }  
 }  
  
 */\*\*  
 \* Clean the words -- remove paragraphs, split hyphens and remove links  
 \*   
 \** ***@param seperatedWords*** *\* the raw words from the input file  
 \*/* **private void** tokenizer(String[] seperatedWords) {  
 **for** (String word : seperatedWords) {  
 word = word.replace(**"\r"**, **""**);  
 word = word.replace(**"\n"**, **""**);  
 **if** (word.length() != 0 && !word.isEmpty() && !word.equals(**" "**)  
 && !word.equals(**""**)) {  
  
 **if** (word.contains(**"http://"**) || word.contains(**"www."**)) {  
 processURL(word);  
 } **else** {  
 **if** (word.contains(**"-"**)) {  
 String[] splitWords = splitHyphenWord(word);  
 **for** (String string : splitWords) {  
 **allWords**.add(cleanString(string));  
 }  
 } **else** {  
 **allWords**.add(cleanString(word));  
 }  
 }  
 }  
 }  
  
 finalPassToSplitWordsWithAttachedNumbers();  
 }  
  
 **private void** tokenizer(ArrayList<String> seperatedWords) {  
 **for** (String word : seperatedWords) {  
 word = word.replace(**"\r"**, **""**);  
 word = word.replace(**"\n"**, **""**);  
 **if** (word.length() != 0 && !word.isEmpty() && !word.equals(**" "**)  
 && !word.equals(**""**)) {  
  
 **if** (word.contains(**"http://"**) || word.contains(**"www."**)) {  
 processURL(word);  
 } **else** {  
 **if** (word.contains(**"-"**)) {  
 String[] splitWords = splitHyphenWord(word);  
 **for** (String string : splitWords) {  
 **allWords**.add(cleanString(string));  
 }  
 } **else** {  
 **allWords**.add(cleanString(word));  
 }  
 }  
 }  
 }  
  
 *//finalPassToSplitWordsWithAttachedNumbers();* }  
  
 */\*\*  
 \* If a number has been attached to a word, it must be separated. The  
 \* separated word is inserted, in order, into allWords  
 \*/* **private void** finalPassToSplitWordsWithAttachedNumbers() {  
 *// add(int index, E element)* **for** (**int** i = 0; i < **allWords**.size(); i++) {  
 String wordToTest = **allWords**.get(i);  
 **int** index = containsLettersAndNumbers(wordToTest);  
  
 **if** (index >= 0) { *//if less than 0, no digit was found* {  
 String[] word = **allWords**.get(i).split(  
 **"(?<=\\D)(?=\\d)|(?<=\\d)(?=\\D)"**);  
 **allWords**.set(i, word[0]);  
 **allWords**.add(i + 1, word[1]);  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* This is a helper method for finalPassToRemoveAnyAttachedNumbers()  
 \*   
 \** ***@param wordToTest*** *\* A candidate word to split by a digit  
 \** ***@return*** *The index of the digit (where to split)  
 \*/* **private int** containsLettersAndNumbers(String wordToTest) {  
  
 ArrayList<Integer> indexes = **new** ArrayList<Integer>();  
 **boolean** letterFound = **false**;  
  
 **for** (**int** i = 0; i < wordToTest.length(); i++) {  
 **if** (!Character.*isLetter*(wordToTest.charAt(i))) {  
 indexes.add(i);  
 }  
 **if** (Character.*isLetter*(wordToTest.charAt(i))) {  
 letterFound = **true**;  
 }  
 }  
  
 **if** (!indexes.isEmpty() && letterFound) {  
 **return** indexes.get(0);  
 }  
  
 **return** -1;  
  
 }  
  
 **private void** processURL(String URL) {  
 URL = URL.replaceAll(**"\\W"**, **" "**);  
 URL = URL.replaceAll(**"\_"**, **" "**);  
  
 String[] URLs = URL.split(**" "**);  
  
 **for** (String string : URLs) {  
 **if** (string.length() != 0)  
 **allWords**.add(cleanString(string));  
 }  
 }  
  
 */\*\*  
 \* Removes all instances of stop words from the allWords array  
 \*   
 \** ***@param stopWords*** *\* the stop words from the stopWord input file  
 \*/* **private void** removeStopWords(String[] stopWords) {  
  
 **for** (String string : stopWords) {  
 **this**.**stopWords**.add(string);  
 }  
  
 **for** (**int** j = 0; j < stopWords.**length**; j++) {  
 **for** (**int** i = 0; i < **allWords**.size(); i++) {  
 **if** (**allWords**.get(i).equals(**this**.**stopWords**.get(j))) {  
 **allWords**.remove(i);  
 }  
 }  
 }  
  
 }  
  
 */\*\*  
 \** ***@param word*** *\* a "word" containing a hyphen to be split  
 \** ***@return*** *an array of 2 words after being split from the array  
 \*/* **private** String[] splitHyphenWord(String word) {  
 String[] splitWords = **null**;  
 **if** (word.contains(**"-"**)) {  
 splitWords = word.split(**"-"**);  
  
 }  
 **return** splitWords;  
 }  
  
 */\*\*  
 \* Using regex, remove numbers and non-alphanumeric chars, make lower case  
 \* and remove trailing spaces  
 \*   
 \** ***@param word*** *\* the word to "clean"  
 \** ***@return*** *a "cleaned" word  
 \*/* **private** String cleanString(String word) {  
 *// word = word.replaceAll("[0-9]", "");* word = word.replaceAll(**"\\W"**, **""**);  
 **return** word.toLowerCase().trim();  
 }  
  
 */\*\*  
 \* If any empty words have made it into the list this will remove them  
 \*/* **private void** removeEmptyWords() {  
 **for** (**int** i = 0; i < **allWords**.size(); i++) {  
 **if** (**allWords**.get(i).length() == 0) {  
 **allWords**.remove(i);  
 }  
 }  
 }  
}

**کلاس EnglishStopWords:**

**public class** EnglishStopWords {  
 **public static** String[] *stopWords* = { **"a"**, **"about"**, **"above"**, **"after"**, **"again"**, **"against"**, **"ain"**, **"all"**, **"am"**, **"an"**,  
 **"and"**, **"any"**, **"are"**, **"aren"**, **"aren't"**, **"as"**, **"at"**, **"be"**, **"because"**, **"been"**, **"before"**, **"being"**, **"below"**,  
 **"between"**, **"both"**, **"but"**, **"by"**, **"can"**, **"couldn"**, **"couldn't"**, **"d"**, **"did"**, **"didn"**, **"didn't"**, **"do"**, **"does"**,  
 **"doesn"**, **"doesn't"**, **"doing"**, **"don"**, **"don't"**, **"down"**, **"during"**, **"each"**, **"few"**, **"for"**, **"from"**, **"further"**,  
 **"had"**, **"hadn"**, **"hadn't"**, **"has"**, **"hasn"**, **"hasn't"**, **"have"**, **"haven"**, **"haven't"**, **"having"**, **"he"**, **"her"**, **"here"**,  
 **"hers"**, **"herself"**, **"him"**, **"himself"**, **"his"**, **"how"**, **"i"**, **"if"**, **"in"**, **"into"**, **"is"**, **"isn"**, **"isn't"**, **"it"**,  
 **"it's"**, **"its"**, **"itself"**, **"just"**, **"ll"**, **"m"**, **"ma"**, **"me"**, **"mightn"**, **"mightn't"**, **"more"**, **"most"**, **"mustn"**,  
 **"mustn't"**, **"my"**, **"myself"**, **"needn"**, **"needn't"**, **"no"**, **"nor"**, **"not"**, **"now"**, **"o"**, **"of"**, **"off"**, **"on"**, **"once"**,  
 **"only"**, **"or"**, **"other"**, **"our"**, **"ours"**, **"ourselves"**, **"out"**, **"over"**, **"own"**, **"re"**, **"s"**, **"same"**, **"shan"**,  
 **"shan't"**, **"she"**, **"she's"**, **"should"**, **"should've"**, **"shouldn"**, **"shouldn't"**, **"so"**, **"some"**, **"such"**, **"t"**, **"than"**,  
 **"that"**, **"that'll"**, **"the"**, **"their"**, **"theirs"**, **"them"**, **"themselves"**, **"then"**, **"there"**, **"these"**, **"they"**, **"this"**,  
 **"those"**, **"through"**, **"to"**, **"too"**, **"under"**, **"until"**, **"up"**, **"ve"**, **"very"**, **"was"**, **"wasn"**, **"wasn't"**, **"we"**,  
 **"were"**, **"weren"**, **"weren't"**, **"what"**, **"when"**, **"where"**, **"which"**, **"while"**, **"who"**, **"whom"**, **"why"**, **"will"**, **"with"**,  
 **"won"**, **"won't"**, **"wouldn"**, **"wouldn't"**, **"y"**, **"you"**, **"you'd"**, **"you'll"**, **"you're"**, **"you've"**, **"your"**, **"yours"**,  
 **"yourself"**, **"yourselves"**, **"could"**, **"he'd"**, **"he'll"**, **"he's"**, **"here's"**, **"how's"**, **"i'd"**, **"i'll"**, **"i'm"**, **"i've"**,  
 **"let's"**, **"ought"**, **"she'd"**, **"she'll"**, **"that's"**, **"there's"**, **"they'd"**, **"they'll"**, **"they're"**, **"they've"**, **"we'd"**,  
 **"we'll"**, **"we're"**, **"we've"**, **"what's"**, **"when's"**, **"where's"**, **"who's"**, **"why's"**, **"would"**, **"able"**, **"abst"**,  
 **"accordance"**, **"according"**, **"accordingly"**, **"across"**, **"act"**, **"actually"**, **"added"**, **"adj"**, **"affected"**,  
 **"affecting"**, **"affects"**, **"afterwards"**, **"ah"**, **"almost"**, **"alone"**, **"along"**, **"already"**, **"also"**, **"although"**,  
 **"always"**, **"among"**, **"amongst"**, **"announce"**, **"another"**, **"anybody"**, **"anyhow"**, **"anymore"**, **"anyone"**, **"anything"**,  
 **"anyway"**, **"anyways"**, **"anywhere"**, **"apparently"**, **"approximately"**, **"arent"**, **"arise"**, **"around"**, **"aside"**, **"ask"**,  
 **"asking"**, **"auth"**, **"available"**, **"away"**, **"awfully"**, **"b"**, **"back"**, **"became"**, **"become"**, **"becomes"**, **"becoming"**,  
 **"beforehand"**, **"begin"**, **"beginning"**, **"beginnings"**, **"begins"**, **"behind"**, **"believe"**, **"beside"**, **"besides"**,  
 **"beyond"**, **"biol"**, **"brief"**, **"briefly"**, **"c"**, **"ca"**, **"came"**, **"cannot"**, **"can't"**, **"cause"**, **"causes"**, **"certain"**,  
 **"certainly"**, **"co"**, **"com"**, **"come"**, **"comes"**, **"contain"**, **"containing"**, **"contains"**, **"couldnt"**, **"date"**,  
 **"different"**, **"done"**, **"downwards"**, **"due"**, **"e"**, **"ed"**, **"edu"**, **"effect"**, **"eg"**, **"eight"**, **"eighty"**, **"either"**,  
 **"else"**, **"elsewhere"**, **"end"**, **"ending"**, **"enough"**, **"especially"**, **"et"**, **"etc"**, **"even"**, **"ever"**, **"every"**,  
 **"everybody"**, **"everyone"**, **"everything"**, **"everywhere"**, **"ex"**, **"except"**, **"f"**, **"far"**, **"ff"**, **"fifth"**, **"first"**,  
 **"five"**, **"fix"**, **"followed"**, **"following"**, **"follows"**, **"former"**, **"formerly"**, **"forth"**, **"found"**, **"four"**,  
 **"furthermore"**, **"g"**, **"gave"**, **"get"**, **"gets"**, **"getting"**, **"give"**, **"given"**, **"gives"**, **"giving"**, **"go"**, **"goes"**,  
 **"gone"**, **"got"**, **"gotten"**, **"h"**, **"happens"**, **"hardly"**, **"hed"**, **"hence"**, **"hereafter"**, **"hereby"**, **"herein"**, **"heres"**,  
 **"hereupon"**, **"hes"**, **"hi"**, **"hid"**, **"hither"**, **"home"**, **"howbeit"**, **"however"**, **"hundred"**, **"id"**, **"ie"**, **"im"**,  
 **"immediate"**, **"immediately"**, **"importance"**, **"important"**, **"inc"**, **"indeed"**, **"index"**, **"information"**, **"instead"**,  
 **"invention"**, **"inward"**, **"itd"**, **"it'll"**, **"j"**, **"k"**, **"keep"**, **"keeps"**, **"kept"**, **"kg"**, **"km"**, **"know"**, **"known"**,  
 **"knows"**, **"l"**, **"largely"**, **"last"**, **"lately"**, **"later"**, **"latter"**, **"latterly"**, **"least"**, **"less"**, **"lest"**, **"let"**,  
 **"lets"**, **"like"**, **"liked"**, **"likely"**, **"line"**, **"little"**, **"'ll"**, **"look"**, **"looking"**, **"looks"**, **"ltd"**, **"made"**,  
 **"mainly"**, **"make"**, **"makes"**, **"many"**, **"may"**, **"maybe"**, **"mean"**, **"means"**, **"meantime"**, **"meanwhile"**, **"merely"**, **"mg"**,  
 **"might"**, **"million"**, **"miss"**, **"ml"**, **"moreover"**, **"mostly"**, **"mr"**, **"mrs"**, **"much"**, **"mug"**, **"must"**, **"n"**, **"na"**,  
 **"name"**, **"namely"**, **"nay"**, **"nd"**, **"near"**, **"nearly"**, **"necessarily"**, **"necessary"**, **"need"**, **"needs"**, **"neither"**,  
 **"never"**, **"nevertheless"**, **"new"**, **"next"**, **"nine"**, **"ninety"**, **"nobody"**, **"non"**, **"none"**, **"nonetheless"**, **"noone"**,  
 **"normally"**, **"nos"**, **"noted"**, **"nothing"**, **"nowhere"**, **"obtain"**, **"obtained"**, **"obviously"**, **"often"**, **"oh"**, **"ok"**,  
 **"okay"**, **"old"**, **"omitted"**, **"one"**, **"ones"**, **"onto"**, **"ord"**, **"others"**, **"otherwise"**, **"outside"**, **"overall"**,  
 **"owing"**, **"p"**, **"page"**, **"pages"**, **"part"**, **"particular"**, **"particularly"**, **"past"**, **"per"**, **"perhaps"**, **"placed"**,  
 **"please"**, **"plus"**, **"poorly"**, **"possible"**, **"possibly"**, **"potentially"**, **"pp"**, **"predominantly"**, **"present"**,  
 **"previously"**, **"primarily"**, **"probably"**, **"promptly"**, **"proud"**, **"provides"**, **"put"**, **"q"**, **"que"**, **"quickly"**,  
 **"quite"**, **"qv"**, **"r"**, **"ran"**, **"rather"**, **"rd"**, **"readily"**, **"really"**, **"recent"**, **"recently"**, **"ref"**, **"refs"**,  
 **"regarding"**, **"regardless"**, **"regards"**, **"related"**, **"relatively"**, **"research"**, **"respectively"**, **"resulted"**,  
 **"resulting"**, **"results"**, **"right"**, **"run"**, **"said"**, **"saw"**, **"say"**, **"saying"**, **"says"**, **"sec"**, **"section"**, **"see"**,  
 **"seeing"**, **"seem"**, **"seemed"**, **"seeming"**, **"seems"**, **"seen"**, **"self"**, **"selves"**, **"sent"**, **"seven"**, **"several"**,  
 **"shall"**, **"shed"**, **"shes"**, **"show"**, **"showed"**, **"shown"**, **"showns"**, **"shows"**, **"significant"**, **"significantly"**,  
 **"similar"**, **"similarly"**, **"since"**, **"six"**, **"slightly"**, **"somebody"**, **"somehow"**, **"someone"**, **"somethan"**,  
 **"something"**, **"sometime"**, **"sometimes"**, **"somewhat"**, **"somewhere"**, **"soon"**, **"sorry"**, **"specifically"**, **"specified"**,  
 **"specify"**, **"specifying"**, **"still"**, **"stop"**, **"strongly"**, **"sub"**, **"substantially"**, **"successfully"**,  
 **"sufficiently"**, **"suggest"**, **"sup"**, **"sure"**, **"take"**, **"taken"**, **"taking"**, **"tell"**, **"tends"**, **"th"**, **"thank"**,  
 **"thanks"**, **"thanx"**, **"thats"**, **"that've"**, **"thence"**, **"thereafter"**, **"thereby"**, **"thered"**, **"therefore"**, **"therein"**,  
 **"there'll"**, **"thereof"**, **"therere"**, **"theres"**, **"thereto"**, **"thereupon"**, **"there've"**, **"theyd"**, **"theyre"**, **"think"**,  
 **"thou"**, **"though"**, **"thoughh"**, **"thousand"**, **"throug"**, **"throughout"**, **"thru"**, **"thus"**, **"til"**, **"tip"**, **"together"**,  
 **"took"**, **"toward"**, **"towards"**, **"tried"**, **"tries"**, **"truly"**, **"try"**, **"trying"**, **"ts"**, **"twice"**, **"two"**, **"u"**, **"un"**,  
 **"unfortunately"**, **"unless"**, **"unlike"**, **"unlikely"**, **"unto"**, **"upon"**, **"ups"**, **"us"**, **"use"**, **"used"**, **"useful"**,  
 **"usefully"**, **"usefulness"**, **"uses"**, **"using"**, **"usually"**, **"v"**, **"value"**, **"various"**, **"'ve"**, **"via"**, **"viz"**, **"vol"**,  
 **"vols"**, **"vs"**, **"w"**, **"want"**, **"wants"**, **"wasnt"**, **"way"**, **"wed"**, **"welcome"**, **"went"**, **"werent"**, **"whatever"**,  
 **"what'll"**, **"whats"**, **"whence"**, **"whenever"**, **"whereafter"**, **"whereas"**, **"whereby"**, **"wherein"**, **"wheres"**,  
 **"whereupon"**, **"wherever"**, **"whether"**, **"whim"**, **"whither"**, **"whod"**, **"whoever"**, **"whole"**, **"who'll"**, **"whomever"**,  
 **"whos"**, **"whose"**, **"widely"**, **"willing"**, **"wish"**, **"within"**, **"without"**, **"wont"**, **"words"**, **"world"**, **"wouldnt"**,  
 **"www"**, **"x"**, **"yes"**, **"yet"**, **"youd"**, **"youre"**, **"z"**, **"zero"**, **"a's"**, **"ain't"**, **"allow"**, **"allows"**, **"apart"**,  
 **"appear"**, **"appreciate"**, **"appropriate"**, **"associated"**, **"best"**, **"better"**, **"c'mon"**, **"c's"**, **"cant"**, **"changes"**,  
 **"clearly"**, **"concerning"**, **"consequently"**, **"consider"**, **"considering"**, **"corresponding"**, **"course"**, **"currently"**,  
 **"definitely"**, **"described"**, **"despite"**, **"entirely"**, **"exactly"**, **"example"**, **"going"**, **"greetings"**, **"hello"**,  
 **"help"**, **"hopefully"**, **"ignored"**, **"inasmuch"**, **"indicate"**, **"indicated"**, **"indicates"**, **"inner"**, **"insofar"**,  
 **"it'd"**, **"keep"**, **"keeps"**, **"novel"**, **"presumably"**, **"reasonably"**, **"second"**, **"secondly"**, **"sensible"**, **"serious"**,  
 **"seriously"**, **"sure"**, **"t's"**, **"third"**, **"thorough"**, **"thoroughly"**, **"three"**, **"well"**, **"wonder"**, **"a"**, **"about"**,  
 **"above"**, **"above"**, **"across"**, **"after"**, **"afterwards"**, **"again"**, **"against"**, **"all"**, **"almost"**, **"alone"**, **"along"**,  
 **"already"**, **"also"**, **"although"**, **"always"**, **"am"**, **"among"**, **"amongst"**, **"amoungst"**, **"amount"**, **"an"**, **"and"**,  
 **"another"**, **"any"**, **"anyhow"**, **"anyone"**, **"anything"**, **"anyway"**, **"anywhere"**, **"are"**, **"around"**, **"as"**, **"at"**, **"back"**,  
 **"be"**, **"became"**, **"because"**, **"become"**, **"becomes"**, **"becoming"**, **"been"**, **"before"**, **"beforehand"**, **"behind"**,  
 **"being"**, **"below"**, **"beside"**, **"besides"**, **"between"**, **"beyond"**, **"bill"**, **"both"**, **"bottom"**, **"but"**, **"by"**, **"call"**,  
 **"can"**, **"cannot"**, **"cant"**, **"co"**, **"con"**, **"could"**, **"couldnt"**, **"cry"**, **"de"**, **"describe"**, **"detail"**, **"do"**, **"done"**,  
 **"down"**, **"due"**, **"during"**, **"each"**, **"eg"**, **"eight"**, **"either"**, **"eleven"**, **"else"**, **"elsewhere"**, **"empty"**, **"enough"**,  
 **"etc"**, **"even"**, **"ever"**, **"every"**, **"everyone"**, **"everything"**, **"everywhere"**, **"except"**, **"few"**, **"fifteen"**, **"fify"**,  
 **"fill"**, **"find"**, **"fire"**, **"first"**, **"five"**, **"for"**, **"former"**, **"formerly"**, **"forty"**, **"found"**, **"four"**, **"from"**,  
 **"front"**, **"full"**, **"further"**, **"get"**, **"give"**, **"go"**, **"had"**, **"has"**, **"hasnt"**, **"have"**, **"he"**, **"hence"**, **"her"**,  
 **"here"**, **"hereafter"**, **"hereby"**, **"herein"**, **"hereupon"**, **"hers"**, **"herself"**, **"him"**, **"himself"**, **"his"**, **"how"**,  
 **"however"**, **"hundred"**, **"ie"**, **"if"**, **"in"**, **"inc"**, **"indeed"**, **"interest"**, **"into"**, **"is"**, **"it"**, **"its"**, **"itself"**,  
 **"keep"**, **"last"**, **"latter"**, **"latterly"**, **"least"**, **"less"**, **"ltd"**, **"made"**, **"many"**, **"may"**, **"me"**, **"meanwhile"**,  
 **"might"**, **"mill"**, **"mine"**, **"more"**, **"moreover"**, **"most"**, **"mostly"**, **"move"**, **"much"**, **"must"**, **"my"**, **"myself"**,  
 **"name"**, **"namely"**, **"neither"**, **"never"**, **"nevertheless"**, **"next"**, **"nine"**, **"no"**, **"nobody"**, **"none"**, **"noone"**,  
 **"nor"**, **"not"**, **"nothing"**, **"now"**, **"nowhere"**, **"of"**, **"off"**, **"often"**, **"on"**, **"once"**, **"one"**, **"only"**, **"onto"**, **"or"**,  
 **"other"**, **"others"**, **"otherwise"**, **"our"**, **"ours"**, **"ourselves"**, **"out"**, **"over"**, **"own"**, **"part"**, **"per"**, **"perhaps"**,  
 **"please"**, **"put"**, **"rather"**, **"re"**, **"same"**, **"see"**, **"seem"**, **"seemed"**, **"seeming"**, **"seems"**, **"serious"**, **"several"**,  
 **"she"**, **"should"**, **"show"**, **"side"**, **"since"**, **"sincere"**, **"six"**, **"sixty"**, **"so"**, **"some"**, **"somehow"**, **"someone"**,  
 **"something"**, **"sometime"**, **"sometimes"**, **"somewhere"**, **"still"**, **"such"**, **"system"**, **"take"**, **"ten"**, **"than"**, **"that"**,  
 **"the"**, **"their"**, **"them"**, **"themselves"**, **"then"**, **"thence"**, **"there"**, **"thereafter"**, **"thereby"**, **"therefore"**,  
 **"therein"**, **"thereupon"**, **"these"**, **"they"**, **"thickv"**, **"thin"**, **"third"**, **"this"**, **"those"**, **"though"**, **"three"**,  
 **"through"**, **"throughout"**, **"thru"**, **"thus"**, **"to"**, **"together"**, **"too"**, **"top"**, **"toward"**, **"towards"**, **"twelve"**,  
 **"twenty"**, **"two"**, **"un"**, **"under"**, **"until"**, **"up"**, **"upon"**, **"us"**, **"very"**, **"via"**, **"was"**, **"we"**, **"well"**, **"were"**,  
 **"what"**, **"whatever"**, **"when"**, **"whence"**, **"whenever"**, **"where"**, **"whereafter"**, **"whereas"**, **"whereby"**, **"wherein"**,  
 **"whereupon"**, **"wherever"**, **"whether"**, **"which"**, **"while"**, **"whither"**, **"who"**, **"whoever"**, **"whole"**, **"whom"**, **"whose"**,  
 **"why"**, **"will"**, **"with"**, **"within"**, **"without"**, **"would"**, **"yet"**, **"you"**, **"your"**, **"yours"**, **"yourself"**,  
 **"yourselves"**, **"the"**, **"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"g"**, **"h"**, **"i"**, **"j"**, **"k"**, **"l"**, **"m"**, **"n"**, **"o"**, **"p"**, **"q"**,  
 **"r"**, **"s"**, **"t"**, **"u"**, **"v"**, **"w"**, **"x"**, **"y"**, **"z"**, **"A"**, **"B"**, **"C"**, **"D"**, **"E"**, **"F"**, **"G"**, **"H"**, **"I"**, **"J"**, **"K"**, **"L"**,  
 **"M"**, **"N"**, **"O"**, **"P"**, **"Q"**, **"R"**, **"S"**, **"T"**, **"U"**, **"V"**, **"W"**, **"X"**, **"Y"**, **"Z"**, **"co"**, **"op"**, **"research-articl"**,  
 **"pagecount"**, **"cit"**, **"ibid"**, **"les"**, **"le"**, **"au"**, **"que"**, **"est"**, **"pas"**, **"vol"**, **"el"**, **"los"**, **"pp"**, **"u201d"**,  
 **"well-b"**, **"http"**, **"volumtype"**, **"par"**, **"0o"**, **"0s"**, **"3a"**, **"3b"**, **"3d"**, **"6b"**, **"6o"**, **"a1"**, **"a2"**, **"a3"**, **"a4"**,  
 **"ab"**, **"ac"**, **"ad"**, **"ae"**, **"af"**, **"ag"**, **"aj"**, **"al"**, **"an"**, **"ao"**, **"ap"**, **"ar"**, **"av"**, **"aw"**, **"ax"**, **"ay"**, **"az"**, **"b1"**,  
 **"b2"**, **"b3"**, **"ba"**, **"bc"**, **"bd"**, **"be"**, **"bi"**, **"bj"**, **"bk"**, **"bl"**, **"bn"**, **"bp"**, **"br"**, **"bs"**, **"bt"**, **"bu"**, **"bx"**, **"c1"**,  
 **"c2"**, **"c3"**, **"cc"**, **"cd"**, **"ce"**, **"cf"**, **"cg"**, **"ch"**, **"ci"**, **"cj"**, **"cl"**, **"cm"**, **"cn"**, **"cp"**, **"cq"**, **"cr"**, **"cs"**, **"ct"**,  
 **"cu"**, **"cv"**, **"cx"**, **"cy"**, **"cz"**, **"d2"**, **"da"**, **"dc"**, **"dd"**, **"de"**, **"df"**, **"di"**, **"dj"**, **"dk"**, **"dl"**, **"do"**, **"dp"**, **"dr"**,  
 **"ds"**, **"dt"**, **"du"**, **"dx"**, **"dy"**, **"e2"**, **"e3"**, **"ea"**, **"ec"**, **"ed"**, **"ee"**, **"ef"**, **"ei"**, **"ej"**, **"el"**, **"em"**, **"en"**, **"eo"**,  
 **"ep"**, **"eq"**, **"er"**, **"es"**, **"et"**, **"eu"**, **"ev"**, **"ex"**, **"ey"**, **"f2"**, **"fa"**, **"fc"**, **"ff"**, **"fi"**, **"fj"**, **"fl"**, **"fn"**, **"fo"**,  
 **"fr"**, **"fs"**, **"ft"**, **"fu"**, **"fy"**, **"ga"**, **"ge"**, **"gi"**, **"gj"**, **"gl"**, **"go"**, **"gr"**, **"gs"**, **"gy"**, **"h2"**, **"h3"**, **"hh"**, **"hi"**,  
 **"hj"**, **"ho"**, **"hr"**, **"hs"**, **"hu"**, **"hy"**, **"i"**, **"i2"**, **"i3"**, **"i4"**, **"i6"**, **"i7"**, **"i8"**, **"ia"**, **"ib"**, **"ic"**, **"ie"**, **"ig"**,  
 **"ih"**, **"ii"**, **"ij"**, **"il"**, **"in"**, **"io"**, **"ip"**, **"iq"**, **"ir"**, **"iv"**, **"ix"**, **"iy"**, **"iz"**, **"jj"**, **"jr"**, **"js"**, **"jt"**, **"ju"**,  
 **"ke"**, **"kg"**, **"kj"**, **"km"**, **"ko"**, **"l2"**, **"la"**, **"lb"**, **"lc"**, **"lf"**, **"lj"**, **"ln"**, **"lo"**, **"lr"**, **"ls"**, **"lt"**, **"m2"**, **"ml"**,  
 **"mn"**, **"mo"**, **"ms"**, **"mt"**, **"mu"**, **"n2"**, **"nc"**, **"nd"**, **"ne"**, **"ng"**, **"ni"**, **"nj"**, **"nl"**, **"nn"**, **"nr"**, **"ns"**, **"nt"**, **"ny"**,  
 **"oa"**, **"ob"**, **"oc"**, **"od"**, **"of"**, **"og"**, **"oi"**, **"oj"**, **"ol"**, **"om"**, **"on"**, **"oo"**, **"oq"**, **"or"**, **"os"**, **"ot"**, **"ou"**, **"ow"**,  
 **"ox"**, **"oz"**, **"p1"**, **"p2"**, **"p3"**, **"pc"**, **"pd"**, **"pe"**, **"pf"**, **"ph"**, **"pi"**, **"pj"**, **"pk"**, **"pl"**, **"pm"**, **"pn"**, **"po"**, **"pq"**,  
 **"pr"**, **"ps"**, **"pt"**, **"pu"**, **"py"**, **"qj"**, **"qu"**, **"r2"**, **"ra"**, **"rc"**, **"rd"**, **"rf"**, **"rh"**, **"ri"**, **"rj"**, **"rl"**, **"rm"**, **"rn"**,  
 **"ro"**, **"rq"**, **"rr"**, **"rs"**, **"rt"**, **"ru"**, **"rv"**, **"ry"**, **"s2"**, **"sa"**, **"sc"**, **"sd"**, **"se"**, **"sf"**, **"si"**, **"sj"**, **"sl"**, **"sm"**,  
 **"sn"**, **"sp"**, **"sq"**, **"sr"**, **"ss"**, **"st"**, **"sy"**, **"sz"**, **"t1"**, **"t2"**, **"t3"**, **"tb"**, **"tc"**, **"td"**, **"te"**, **"tf"**, **"th"**, **"ti"**,  
 **"tj"**, **"tl"**, **"tm"**, **"tn"**, **"tp"**, **"tq"**, **"tr"**, **"ts"**, **"tt"**, **"tv"**, **"tx"**, **"ue"**, **"ui"**, **"uj"**, **"uk"**, **"um"**, **"un"**, **"uo"**,  
 **"ur"**, **"ut"**, **"va"**, **"wa"**, **"vd"**, **"wi"**, **"vj"**, **"vo"**, **"wo"**, **"vq"**, **"vt"**, **"vu"**, **"x1"**, **"x2"**, **"x3"**, **"xf"**, **"xi"**, **"xj"**,  
 **"xk"**, **"xl"**, **"xn"**, **"xo"**, **"xs"**, **"xt"**, **"xv"**, **"xx"**, **"y2"**, **"yj"**, **"yl"**, **"yr"**, **"ys"**, **"yt"**, **"zi"**, **"zz"** };  
}

**کلاس DataAccess:**

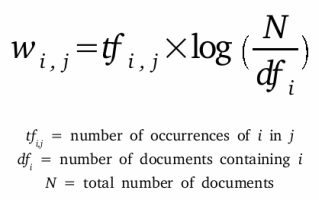
**package** Indexer;  
  
**import** java.sql.\*;  
  
**public class** DataAccess {  
 **private static** DataAccess *access*;  
  
 **private** PreparedStatement **getRightUrlsStatement**;  
 **private** PreparedStatement[] **resetIndexingTablesStatements**;  
 **private** PreparedStatement **getUrlIdStatement**;  
 **private** PreparedStatement **insertDocSizeStatement**;  
 **private** PreparedStatement **findWordIdStatement**;  
 **private** PreparedStatement **isAlreadyWordAddedToUrlStatement**;  
 **private** PreparedStatement **increaseWordFreqStatement**;  
 **private** PreparedStatement **addWordStatement**;  
 **private** PreparedStatement **addWord2Statement**;  
 PreparedStatement **isAlreadyInsertedInDocSizeStatement**;  
  
 **private** DataAccess() **throws** SQLException {  
 init();  
 }  
  
 **public static** Connection getConnection() {  
 Connection connection = **null**;  
 **try** {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 connection = DriverManager.*getConnection*(  
 **"jdbc:mysql://localhost:3306/search\_engine"**, **"root"**, **""**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** connection;  
 }  
  
 **public static** DataAccess getAccess() **throws** SQLException {  
 **if** (*access* == **null**) {  
 *access* = **new** DataAccess();  
 }  
 **return** *access*;  
 }  
  
 **private void** init() **throws** SQLException {  
 Connection dbConnection = *getConnection*();  
 **getRightUrlsStatement** = dbConnection.prepareStatement(**"SELECT \* FROM urls WHERE is\_index=TRUE;"**);  
 **resetIndexingTablesStatements** = **new** PreparedStatement[]{  
 dbConnection.prepareStatement(**"DELETE FROM words;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE words AUTO\_INCREMENT = 1;"**),  
 dbConnection.prepareStatement(**"DELETE FROM word\_doc;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE word\_doc AUTO\_INCREMENT = 1;"**),  
 dbConnection.prepareStatement(**"DELETE FROM doc\_size;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE doc\_size AUTO\_INCREMENT = 1;"**)  
 };  
 **getUrlIdStatement** = dbConnection.prepareStatement(**"SELECT id FROM urls WHERE url=?;"**);  
 **insertDocSizeStatement** = dbConnection.prepareStatement(**"INSERT INTO doc\_size (doc\_url\_id, doc\_size) VALUES (?,?)"**);  
 **findWordIdStatement** = dbConnection.prepareStatement(**"SELECT id FROM words WHERE word=?;"**);  
 **isAlreadyWordAddedToUrlStatement** = dbConnection.prepareStatement(**"SELECT id FROM word\_doc WHERE word\_id=? AND doc\_url\_id=?"**);  
 **increaseWordFreqStatement** = dbConnection.prepareStatement(**"UPDATE word\_doc SET freq=freq+1 WHERE doc\_url\_id=? AND word\_id=?"**);  
 **addWordStatement** = dbConnection.prepareStatement(**"INSERT INTO words (word) VALUES (?);"**);  
 **addWord2Statement** = dbConnection.prepareStatement(**"INSERT INTO word\_doc (word\_id, doc\_url\_id, freq) VALUES (?,?,?);"**);  
 **isAlreadyInsertedInDocSizeStatement** = dbConnection.prepareStatement(**"SELECT \* FROM doc\_size WHERE doc\_url\_id=?"**);  
 }  
  
 **public void** resetIndexingTables() **throws** SQLException {  
 executeMultiStatement(**resetIndexingTablesStatements**);  
 }  
  
 **private void** executeMultiStatement(PreparedStatement[] preparedStatements) **throws** SQLException {  
 **for** (PreparedStatement preparedStatement : preparedStatements) {  
 preparedStatement.execute();  
 }  
 }  
  
 **public** ResultSet getRightUrls() **throws** SQLException {  
 **return getRightUrlsStatement**.executeQuery();  
 }  
  
 **public long** getID(String url) **throws** SQLException {  
 **getUrlIdStatement**.setString(1, url);  
 ResultSet resultSet = **getUrlIdStatement**.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"id"**);  
 }  
  
 **public boolean** insertDocSize(**long** docUrlId, **long** docSize) **throws** SQLException {  
 **insertDocSizeStatement**.setLong(1, docUrlId);  
 **insertDocSizeStatement**.setLong(2, docSize);  
 **return insertDocSizeStatement**.executeUpdate() > 0;  
 }  
  
 **public boolean** isWordAlreadyExisting(String word) **throws** SQLException {  
 **findWordIdStatement**.setString(1, word);  
 **return findWordIdStatement**.executeQuery().next();  
 }  
  
 **public long** getWordId(String word) **throws** SQLException {  
 **findWordIdStatement**.setString(1, word);  
 ResultSet resultSet = **findWordIdStatement**.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"id"**);  
 }  
  
 **public boolean** isAlreadyWordAddedToUrl(**long** wordId, **long** docUrlID) **throws** SQLException {  
 **isAlreadyWordAddedToUrlStatement**.setLong(1, wordId);  
 **isAlreadyWordAddedToUrlStatement**.setLong(2, docUrlID);  
 **return isAlreadyWordAddedToUrlStatement**.executeQuery().next();  
 }  
  
 **public boolean** increaseWordFreq(**long** docUrlId, **long** wordId) **throws** SQLException {  
 **increaseWordFreqStatement**.setLong(1, docUrlId);  
 **increaseWordFreqStatement**.setLong(2, wordId);  
 **return increaseWordFreqStatement**.executeUpdate() > 0;  
 }  
  
 **public boolean** addWord(String word) **throws** SQLException {  
 **addWordStatement**.setString(1, word);  
 **return addWordStatement**.execute();  
 }  
  
 **public boolean** addWord2(**long** wordId, **long** docUrlId, **long** freq) **throws** SQLException {  
 **addWord2Statement**.setLong(1, wordId);  
 **addWord2Statement**.setLong(2, docUrlId);  
 **addWord2Statement**.setLong(3, freq);  
 **return addWord2Statement**.execute();  
 }  
  
 **public boolean** isAlreadyInsertedInDocSize(**long** docUrlId) **throws** SQLException {  
 **isAlreadyInsertedInDocSizeStatement**.setLong(1, docUrlId);  
 **return isAlreadyInsertedInDocSizeStatement**.executeQuery().next();  
 }  
}

**کلاس utill :**

**package** Indexer;  
  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.net.URL;  
**import** java.net.URLConnection;  
**import** java.util.ArrayList;  
  
**public class** utill {  
 **public static void** writeALine() {  
 System.***out***.println(**"--------------------"**);  
 }  
 **public static** ArrayList<String> textToWordsList(String text) {  
 String[] words = text.split(**"[\\s/(,='-]"**);  
 ArrayList<String> arrayList = **new** ArrayList<>();  
 **for** (String item : words) {  
 **if** (!(item.equals(**""**))) arrayList.add(item);  
 }  
 **return** arrayList;  
 }  
 **public static** String getUrlHtml(String url) {  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **try** {  
 URL theUrl = **new** URL(url);  
 URLConnection conn = theUrl.openConnection();  
 BufferedReader br = **new** BufferedReader(  
 **new** InputStreamReader(conn.getInputStream()));  
 String inputLine;  
 **while** ((inputLine = br.readLine()) != **null**) {  
 stringBuilder.append(inputLine);  
 }  
 br.close();  
 } **catch** (Exception ignored) {  
 }  
 **return** stringBuilder.toString();  
 }  
}

**بخش رتبه بندی**

**روش TfIdf**

در این روش رتبه ی لینک ها بر اساس پارامترهای تعداد یک کلمه ی خاص در یک لینک و همه ی لینک ها و تعداد کل لینک به دست می‌آید.

رابطه‌ی TfIdf :

رتبه بندی از طریق این روش در این پروژه با استفاده از دو کلاس TfIdf و DataAccess انجام شده است که هر یک از آن‌ها به ترتیب مورد بررسی قرار می‌گیرند.

کلاس TfIdf : در این کلاس با داشتن ورودی‌های شماره های id کلمه و url می‌توان رتبه مربوط به آن url به ازای یک کلمه را محاسبه کرد. این محاسبه از طریق متدهای tf و idf انجام می‌شود. پس از محاسبه این دو پارامتر در متد سازنده این کلاس محاسبه tf\_idf انجام می گردد و در جدول tf\_idf دیتابیس ذخیره می‌گردد.

کلاس DataAccess : در این کلاس توسط متد getConnection، یک آبجکت از کلاس Connection ایجاد می شود. این کلاس مسئولیت برقراری ارتباط نرم افزار جاوا با دیتابیس MySQL را بر عهده دارد. کوئری های موردنیاز جهت انجام متدهای مربوط به ranking با روش tf\_idf ابتدا در متغیرهای PreparedStatement ذخیره و سپس با متد excute اجرا می‌شوند.

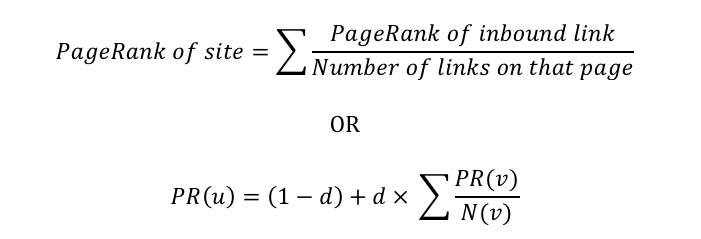
**کلاس TfIdf:**

**package** Ranker.TfIdf;  
  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
  
**public class** TfIdf {  
 **private** DataAccess **dataAccess**;  
 **private long docsSize**;  
 **private long urlId**;  
 **private long wordId**;  
 **private double tfIdf**;  
  
 **private** TfIdf(**long** urlId, **long** wordId) **throws** SQLException, ClassNotFoundException {  
 **this**.**dataAccess** = DataAccess.*getAccess*();  
 **this**.**urlId** = urlId;  
 **this**.**wordId** = wordId;  
 **this**.**docsSize** = **dataAccess**.getDocsSize();  
 **tfIdf** = tf() \* idf();  
 }  
  
 **public static** TfIdf getInstance(**long** urlId, **long** wordId) **throws** SQLException, ClassNotFoundException {  
 **return new** TfIdf(urlId, wordId);  
 }  
  
 **public static void** start() **throws** SQLException, ClassNotFoundException {  
 TfIdf tfIdf;  
 DataAccess dataAccess = DataAccess.*getAccess*();  
 ResultSet resultSet = dataAccess.getRightUrlIds();  
 resultSet.first();  
 ResultSet urlWordIdsResultSet;  
 **while**(**true**){  
 **if** (resultSet.next()) {  
 **do** {  
 **long** urlId = resultSet.getLong(**"id"**);  
 urlWordIdsResultSet = dataAccess.getUrlWordIds(urlId);  
 urlWordIdsResultSet.first();  
 **if** (urlWordIdsResultSet.next()) {  
 **do** {  
 **long** wordId = urlWordIdsResultSet.getLong(**"word\_id"**);  
 tfIdf = TfIdf.*getInstance*(urlId, wordId);  
 tfIdf.rank();  
 System.***out***.println(**"new ranking submitted -> url\_id: "** + urlId + **"\nword\_id: "** + wordId + **"\n\n"**);  
 } **while** (urlWordIdsResultSet.next());  
 }  
 } **while** (resultSet.next());  
 }  
 }  
 }  
  
 **public void** rank() **throws** SQLException {  
 **dataAccess**.insertTfId(**tfIdf**, **urlId**, **wordId**);  
 }  
  
 **public double** tf() **throws** SQLException {  
 **long** wordFreq = **dataAccess**.getWordFreq(**wordId**, **urlId**);  
 **long** docSize = **dataAccess**.getDocSizeByUrlId(**urlId**);  
 **return** (**double**) wordFreq / docSize;  
 }  
  
 **public double** idf() **throws** SQLException {  
 **long** wordFreqInAllDocs = **dataAccess**.getWordFreqInAllDocs(**wordId**);  
 **return** Math.*log*(**docsSize** / wordFreqInAllDocs);  
 }  
}

**کلاس ‌DataAccess :**

**package** Ranker.TfIdf;  
  
**import** java.sql.\*;  
  
**public class** DataAccess {  
 **private static** DataAccess *access*;  
  
 **private** PreparedStatement **insertTfIdfStatement**;  
 **private** PreparedStatement **getDocsSizeStatement**;  
 **private** PreparedStatement **findWordFreqStatement**;  
 **private** PreparedStatement **getDocSizeByUrlIdStatement**;  
 **private** PreparedStatement **getWordFreqInAllDocsStatement**;  
 **private** PreparedStatement **getRightUrlIdsStatement**;  
 **private** PreparedStatement **getUrlWordIdsStatement**;  
 **private** PreparedStatement[] **resetTfIdfTableStatements**;  
  
  
 **private** DataAccess() **throws** SQLException {  
 init();  
 }  
  
 **public static** Connection getConnection() {  
 Connection connection = **null**;  
 **try** {  
 Class.forName(**"com.mysql.jdbc.Driver"**);  
 connection = DriverManager.getConnection(  
 **"jdbc:mysql://localhost:3306/search\_engine"**, **"root"**, **""**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** connection;  
 }  
  
 **public static** DataAccess getAccess() **throws** SQLException, ClassNotFoundException {  
 **if** (access == **null**) {  
 access = **new** DataAccess();  
 }  
 **return** access;  
 }  
  
 **private void** init() **throws** SQLException {  
 Connection dbConnection = getConnection();  
 getDocsSizeStatement = dbConnection.prepareStatement(**"SELECT COUNT(\*) as c from urls"**);  
 insertTfIdfStatement = dbConnection.prepareStatement(**"INSERT INTO tf\_idf (score, doc\_url\_id, word\_id) VALUES (?,?,?)"**);  
 findWordFreqStatement = dbConnection.prepareStatement(**"SELECT freq FROM word\_doc WHERE word\_id=? AND doc\_url\_id=?;"**);  
 getDocSizeByUrlIdStatement = dbConnection.prepareStatement(**"SELECT doc\_size FROM doc\_size WHERE doc\_url\_id=?"**);  
 getWordFreqInAllDocsStatement = dbConnection.prepareStatement(**"SELECT COUNT(\*) AS c FROM word\_doc WHERE word\_id=?"**);  
 getRightUrlIdsStatement = dbConnection.prepareStatement(**"SELECT id FROM urls WHERE is\_index=TRUE"**);  
 getUrlWordIdsStatement = dbConnection.prepareStatement(**"SELECT word\_id FROM word\_doc WHERE doc\_url\_id=?"**);  
 resetTfIdfTableStatements = **new** PreparedStatement[]{  
 dbConnection.prepareStatement(**"DELETE FROM tf\_idf;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE tf\_idf AUTO\_INCREMENT=1"**)  
 };  
 }  
  
 **public long** getDocsSize() **throws** SQLException {  
 ResultSet resultSet = getDocsSizeStatement.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"c"**);  
 }  
  
 **public boolean** insertTfId(**double** score, **long** docUrlId, **long** wordId) **throws** SQLException {  
 insertTfIdfStatement.setDouble(1, score);  
 insertTfIdfStatement.setLong(2, docUrlId);  
 insertTfIdfStatement.setLong(3, wordId);  
 **return** insertTfIdfStatement.execute();  
 }  
  
 **public long** getWordFreq(**long** wordId, **long** docUrlId) **throws** SQLException {  
 findWordFreqStatement.setLong(1, wordId);  
 findWordFreqStatement.setLong(2, docUrlId);  
 ResultSet resultSet = findWordFreqStatement.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"freq"**);  
 }  
  
 **public long** getDocSizeByUrlId(**long** docUrlId) **throws** SQLException {  
 getDocSizeByUrlIdStatement.setLong(1, docUrlId);  
 ResultSet resultSet = getDocSizeByUrlIdStatement.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"doc\_size"**);  
 }  
  
 **public long** getWordFreqInAllDocs(**long** wordId) **throws** SQLException {  
 getWordFreqInAllDocsStatement.setLong(1, wordId);  
 ResultSet resultSet = getWordFreqInAllDocsStatement.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"c"**);  
 }  
  
 **public** ResultSet getRightUrlIds() **throws** SQLException {  
 **return** getRightUrlIdsStatement.executeQuery();  
 }  
  
 **public** ResultSet getUrlWordIds(**long** docUrlId) **throws** SQLException {  
 getUrlWordIdsStatement.setLong(1, docUrlId);  
 **return** getUrlWordIdsStatement.executeQuery();  
 }  
  
 **public void** resetTfIdfTable() **throws** SQLException {  
 executeMultiStatement(resetTfIdfTableStatements);  
 }  
  
 **private void** executeMultiStatement(PreparedStatement[] preparedStatements) **throws** SQLException {  
 **for** (PreparedStatement preparedStatement : preparedStatements) {  
 preparedStatement.execute();  
 }  
 }  
}

**روش PageRank:**

در این روش رتبه بندی بر اساس پارامتر PageRank محاسبه می‌شود. این پارامتر نشان دهنده میزات اعتبار یک لینک بر اساس تعداد مرتبه هایی که به آن در لینک های دیگر آدرس داده شده است. این پارامتر بر اساس رابطه ی ذیل محاسبه می‌گردد.

کلاس PageRank: در این کلاس محاسبه pagerank هر یک از url ها با استفاده از متد calculate انجام می‌شود. در طول اجرای این متد، متد initizlize جهت مقداردهی تعداد لینک های خروجی اجرا می شود. این متد بر اساس اطلاعات جدول links در دیتابیس محاسبه می کند که هر یک از لینک ها چه تعداد لینک خروجی را داشته اند. همچنین محاسبه می گردد که هر یک از لینک ها چند با توسط لینک های دیگر آدرس داده شده اند. مجموعه ی این اطلاعات در متد calculate محاسبه امتیاز pagerank را موجب می‌شود. متد start در یک حلقه ی بی نهایت امتیاز pagerank لینک های موجود در جدول urls را محاسبه و در جدول pagerank ثبت می کند.

کلاس DataAccess : در این کلاس توسط متد getConnection، یک آبجکت از کلاس Connection ایجاد می شود. این کلاس مسئولیت برقراری ارتباط نرم افزار جاوا با دیتابیس MySQL را بر عهده دارد. کوئری های موردنیاز جهت انجام متدهای مربوط به ranking با روش pagerank ابتدا در متغیرهای PreparedStatement ذخیره و سپس با متد excute اجرا می‌شوند.

کلاس Pagerank :

**package** Ranker.PageRank;  
  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
  
**public class** PageRank {  
  
 **private static final double *COMPARE\_DELTA*** = 0.0001;  
 **private static final double *DAMPING\_FACTOR*** = 0.9;  
 **private static final boolean *IS\_CALCULATION\_DAMPED*** = **true**;  
 **private** DataAccess **dbAccess**;  
 **private** Double **equalDistributionValue**;  
 **private long startId**, **endId**;  
  
 */\*  
 connect to database and initialize vektor of every node with 1/amountOfNodes  
 \*/* **public** PageRank(**long** startId, **long** endId) **throws** SQLException, ClassNotFoundException {  
 **this**.**startId** = startId;  
 **this**.**endId** = endId;  
 **dbAccess** = DataAccess.*getAccess*();  
 initialize();  
 }  
  
 **public static** PageRank getInstance(**long** startId, **long** endId) **throws** SQLException, ClassNotFoundException {  
 **return new** PageRank(startId, endId);  
 }  
  
 **public static void** start(**long** limit) **throws** SQLException, ClassNotFoundException {  
 DataAccess dataAccess = DataAccess.*getAccess*();  
 dataAccess.resetPageRankTable();  
 **long** startId, endId;  
 **int** i = 0;  
 **while** (**true**) {  
 **int** chapter = i + 1;  
 System.***out***.println(**"PageRanker (chapter "** + chapter + **") : started!"**);  
 System.***out***.println(**"PageRanker (chapter "** + chapter + **") : loading ..."**);  
 startId = i \* limit;  
 endId = (i + 1) \* limit;  
 **try** {  
 *getInstance*(startId, endId).calculatePageRank();  
 } **catch** (Exception ignored) {  
 }  
 System.***out***.println(**"PageRanker (chapter "** + chapter + **") : finished!"**);  
 i++;  
 }  
 }  
  
 */\*  
 set every vektor at the beginning of the calculation to initial value which is 1/amountOfNodes  
 \*/* **private void** initializeVector() **throws** SQLException {  
 **dbAccess**.initVektor(getEqualDistributionVektorValue(), **startId**, **endId**);  
 }  
  
 **private void** initialize() **throws** SQLException {  
 ResultSet resultSet = **dbAccess**.getUrls(**startId**, **endId**);  
 resultSet.first();  
 **if** (resultSet.next()) {  
 **do** {  
 **long** docUrlID = resultSet.getLong(**"id"**);  
 **dbAccess**.insertUrl(docUrlID);  
 initializeVector();  
 **int** amountOutgoingLinks = getOutgoingLinksCount(docUrlID);  
 **dbAccess**.setOutgoingLinksValue(amountOutgoingLinks, docUrlID, **startId**, **endId**);  
 } **while** (resultSet.next());  
 }  
 }  
  
 */\*  
 looping calculation while pagerank != vektor, every loop is headed by setting vektor = pagerank and pagerank = 0  
 \*/* **public void** calculatePageRank() **throws** SQLException {  
 calculate();  
 **while** (!isCalculationFinished()) {  
 *//System.out.println(dbAccess.getVektorSum());* prepareNextRound();  
 calculate();  
 }  
 }  
  
 */\*  
 count amount of nodes  
 \*/* **public long** getAmountOfNodes() **throws** SQLException {  
*// return dbAccess.getNodesCount(limit);* **return endId** - **startId**;  
 }  
  
 */\*  
 iterate over all nodes, calculate increasing value by dividing vektor by the amount of outgoing links and  
 increase the pagerank value of the nodes of the outgoing links  
 \*/* **public void** calculate() **throws** SQLException {  
 ResultSet nodes = **dbAccess**.getPageRankTable(**startId**, **endId**);  
 **while** (nodes.next()) {  
 **int** id = nodes.getInt(**"doc\_url\_id"**);  
 **double** vektor = nodes.getDouble(**"vektor"**);  
 **int** outgoingLinksCount = nodes.getInt(**"outgoing"**);  
 **double** increasingValue = vektor / outgoingLinksCount;  
 increaseOutgoingLinksByValue(id, increasingValue);  
 }  
 **if** (isCalculationDamped()) dampPageRank();  
 }  
  
 **private void** dampPageRank() **throws** SQLException {  
 **dbAccess**.dampPageRank(***DAMPING\_FACTOR***, **equalDistributionValue**, **startId**, **endId**);  
 }  
  
 */\*  
 get amount of outgoing links for the specified sourceId  
 \*/* **private int** getOutgoingLinksCount(**long** sourceId) **throws** SQLException {  
 **return dbAccess**.countOutgoingLinks(sourceId);  
 }  
  
 */\*  
 get all outgoing links and increase the value of the Pagerank of the connected nodes  
 \*/* **private void** increaseOutgoingLinksByValue(**int** sourceId, **double** value) **throws** SQLException {  
 ResultSet outgoingLink = **dbAccess**.getOutgoingLinks(sourceId);  
  
 **while** (outgoingLink.next()) {  
 **int** targetId = outgoingLink.getInt(**"target"**);  
 increasePagerankByValue(targetId, value);  
 }  
 }  
  
 */\*  
 increases the pageranke of a specified node by the given value  
 \*/* **private void** increasePagerankByValue(**int** targetId, **double** increasingValue) **throws** SQLException {  
 **dbAccess**.increasePageRank(increasingValue, targetId, **startId**, **endId**);  
 }  
  
 */\*  
 prepare the next calculation round by setting vektor to pagerank and pagerank to 0  
 \*/* **private void** prepareNextRound() **throws** SQLException {  
 **dbAccess**.prepareCalculation(**startId**, **endId**);  
 }  
  
 **private boolean** isCalculationDamped() {  
 **return *IS\_CALCULATION\_DAMPED***;  
 }  
  
 */\*  
 check every node if the vektor and pagerank are similar, if not it returns false by the first node which dont fit  
 \*/* **private boolean** isCalculationFinished() **throws** SQLException {  
 ResultSet node = **dbAccess**.getPageRankTable(**startId**, **endId**);  
 **while** (node.next()) {  
 **if** (!isPageRankSimilarToVektor(node)) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 */\*  
 compare vektor and pagerank of given node by given COMPARE\_DELTA  
 \*/* **private boolean** isPageRankSimilarToVektor(ResultSet node) **throws** SQLException {  
 **double** vektor = node.getDouble(**"Vektor"**);  
 **double** pageRank = node.getDouble(**"pagerank"**);  
 **return** !(Math.*abs*(vektor - pageRank) >= ***COMPARE\_DELTA***);  
 }  
  
 **private double** getEqualDistributionVektorValue() **throws** SQLException {  
 **if** (**equalDistributionValue** == **null**) {  
 **equalDistributionValue** = (1.0 / **dbAccess**.getNodesCount());  
 }  
 **return equalDistributionValue**;  
 }  
}

کلاس DataAccess :

**package** Ranker.PageRank;  
  
**import** java.sql.\*;  
  
**public class** DataAccess {  
 **private static** DataAccess *access*;  
  
 **private** PreparedStatement **initVektorStatement**;  
 **private** PreparedStatement[] **resetPageRankTableStatements**;  
 **private** PreparedStatement **getUrlsStatement**;  
 **private** PreparedStatement **setOutgoingStatement**;  
 **private** PreparedStatement **countNodesStatement**;  
 **private** PreparedStatement **getPageRankTableStatement**;  
 **private** PreparedStatement **prepareDampStatement**;  
 **private** PreparedStatement **countOutgoingLinksStatement**;  
 **private** PreparedStatement **getOutgoingLinksStatement**;  
 **private** PreparedStatement **increasePageRankStatement**;  
 **private** PreparedStatement **prepareCalculationStatement**;  
 **private** PreparedStatement **insertUrlStatement**;  
 **private** PreparedStatement **getAllUrlsStatement**;  
  
  
 **private** DataAccess() **throws** SQLException {  
 init();  
 }  
  
 **public static** Connection getConnection() {  
 Connection connection = **null**;  
 **try** {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 connection = DriverManager.*getConnection*(  
 **"jdbc:mysql://localhost:3306/search\_engine"**, **"root"**, **""**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** connection;  
 }  
  
 **public static** DataAccess getAccess() **throws** SQLException {  
 **if** (*access* == **null**) {  
 *access* = **new** DataAccess();  
 }  
 **return** *access*;  
 }  
  
 **private void** init() **throws** SQLException {  
 Connection dbConnection = *getConnection*();  
 **initVektorStatement** = dbConnection.prepareStatement(**"Update pagerank SET vektor=? WHERE doc\_url\_id>=? AND doc\_url\_id<=?;"**);  
 **resetPageRankTableStatements** = **new** PreparedStatement[]{  
 dbConnection.prepareStatement(**"DELETE FROM pagerank;"**),  
 dbConnection.prepareStatement(**"ALTER TABLE pagerank AUTO\_INCREMENT=1;"**)  
 };  
 **getUrlsStatement** = dbConnection.prepareStatement(**"SELECT \* FROM urls WHERE id>=? AND id<=?;"**);  
 **getAllUrlsStatement** = dbConnection.prepareStatement(**"SELECT \* FROM urls;"**);  
 **setOutgoingStatement** = dbConnection.prepareStatement(**"UPDATE pagerank SET outgoing = ? WHERE doc\_url\_id=? AND (doc\_url\_id>=? AND doc\_url\_id<=?);"**);  
 **countNodesStatement** = dbConnection.prepareStatement(**"SELECT Count(\*) AS c FROM urls;"**);  
 **getPageRankTableStatement** = dbConnection.prepareStatement(**"SELECT \* FROM pagerank WHERE doc\_url\_id>=? AND doc\_url\_id<=?;"**);  
 **prepareDampStatement** = dbConnection.prepareStatement(**"UPDATE pagerank SET pagerank=pagerank\*?+?\*? WHERE doc\_url\_id>=? AND doc\_url\_id<=?;"**);  
 **countOutgoingLinksStatement** = dbConnection.prepareStatement(**"SELECT Count(\*) AS c FROM links WHERE source = ?;"**);  
 **getOutgoingLinksStatement** = dbConnection.prepareStatement(**"SELECT \* FROM links WHERE source=?;"**);  
 **increasePageRankStatement** = dbConnection.prepareStatement(**"Update pagerank SET pagerank=pagerank+? WHERE id=? AND (doc\_url\_id>=? AND doc\_url\_id<=?);"**);  
 **prepareCalculationStatement** = dbConnection.prepareStatement(**"Update pagerank SET vektor=pagerank, pagerank=0 WHERE doc\_url\_id>=? AND doc\_url\_id<=?;"**);  
 **insertUrlStatement** = dbConnection.prepareStatement(**"INSERT INTO pagerank (doc\_url\_id) VALUES (?)"**);  
 }  
  
 **public void** resetPageRankTable() **throws** SQLException {  
 executeMultiStatement(**resetPageRankTableStatements**);  
 }  
  
 **private void** executeMultiStatement(PreparedStatement[] preparedStatements) **throws** SQLException {  
 **for** (PreparedStatement preparedStatement : preparedStatements) {  
 preparedStatement.execute();  
 }  
 }  
  
 **public boolean** initVektor(Double equalDistributionVektorValue, **long** startId, **long** endId) **throws** SQLException {  
 **initVektorStatement**.setDouble(1, equalDistributionVektorValue);  
 **initVektorStatement**.setDouble(2, startId);  
 **initVektorStatement**.setDouble(3, endId);  
 **return initVektorStatement**.execute();  
 }  
  
 **public** ResultSet getUrls(**long** startId, **long** endId) **throws** SQLException {  
 **getUrlsStatement**.setLong(1, startId);  
 **getUrlsStatement**.setLong(2, endId);  
 **return getUrlsStatement**.executeQuery();  
 }  
  
 **public** ResultSet getAllUrls() **throws** SQLException {  
 **return getUrlsStatement**.executeQuery();  
 }  
  
 **public** ResultSet getPageRankTable(**long** startId, **long** endId) **throws** SQLException {  
 **getPageRankTableStatement**.setLong(1, startId);  
 **getPageRankTableStatement**.setLong(2, endId);  
 **return getPageRankTableStatement**.executeQuery();  
 }  
  
 **public boolean** setOutgoingLinksValue(**int** outgoing, **long** id,**long** startId,**long** endId) **throws** SQLException {  
 **setOutgoingStatement**.setInt(1, outgoing);  
 **setOutgoingStatement**.setLong(2, id);  
 **setOutgoingStatement**.setLong(3, startId);  
 **setOutgoingStatement**.setLong(4, endId);  
 **return setOutgoingStatement**.execute();  
 }  
  
 **public long** getNodesCount() **throws** SQLException {  
 ResultSet resultSet = **countNodesStatement**.executeQuery();  
 resultSet.first();  
 **return** resultSet.getLong(**"c"**);  
 }  
  
 **public boolean** dampPageRank(**double** dampingFactor, **double** equalDistributionVektorValue, **long** startId, **long** endId) **throws** SQLException {  
 **prepareDampStatement**.setDouble(1, dampingFactor);  
 **prepareDampStatement**.setDouble(2, equalDistributionVektorValue);  
 **prepareDampStatement**.setDouble(3, 1 - dampingFactor);  
 **prepareDampStatement**.setDouble(4, startId);  
 **prepareDampStatement**.setDouble(5, endId);  
 **return prepareDampStatement**.execute();  
 }  
  
 **public int** countOutgoingLinks(**long** sourceID) **throws** SQLException {  
 **countOutgoingLinksStatement**.setLong(1, sourceID);  
 ResultSet resultSet = **countOutgoingLinksStatement**.executeQuery();  
 resultSet.first();  
 **return** resultSet.getInt(**"c"**);  
 }  
  
 **public** ResultSet getOutgoingLinks(**long** sourceID) **throws** SQLException {  
 **getOutgoingLinksStatement**.setLong(1, sourceID);  
 **return getOutgoingLinksStatement**.executeQuery();  
 }  
  
 **public boolean** increasePageRank(**double** increasingValue, **long** targetID, **long** startId, **long** endId) **throws** SQLException {  
 **increasePageRankStatement**.setDouble(1, increasingValue);  
 **increasePageRankStatement**.setLong(2, targetID);  
 **increasePageRankStatement**.setLong(3,startId);  
 **increasePageRankStatement**.setLong(4,endId);  
 **return increasePageRankStatement**.execute();  
 }  
  
 **public boolean** prepareCalculation(**long** startId, **long** endId) **throws** SQLException {  
 **prepareCalculationStatement**.setLong(1,startId);  
 **prepareCalculationStatement**.setLong(2,endId);  
 **return prepareCalculationStatement**.execute();  
 }  
  
 **public boolean** insertUrl(**long** docUrlId) **throws** SQLException {  
 **insertUrlStatement**.setLong(1, docUrlId);  
 **return insertUrlStatement**.execute();  
 }  
  
}

**بخش جستجو**

این بخش با بهره گیری از اطلاعات رتبه های tf\_idf و pagerank در دیتابیس لینک هایی که شامل کلمه ورودی در نرم افزار جستجو باشند را می یابد و به ترتیب رتبه ای که دارند نمایش می دهد.

این بخش با به کارگیری فریمورک laravel که یکی از فریمورک های طراحی وب سایت به زبان php است انجام شده است. جهت استفاده از این بخش ابتدا نرم افزارهای composer و لاراول باید نصب شوند.

برای اجرا از کد ذیل استفاده می شود.

php artisan serve

در بین فایل هایی که در پروژه وب سایت جستجو است دو فایل SearchController.php و index.blade.php کدهای اصلی بخش search را پردازش می‌کنند.

SearchController.php : در این کلاس پس از دریافت کلمه مورد جستجو در متد index آن را ابتدا به حروف کوچک تبدیل کرده و سپس tokenize می کند. بعد از این مرحله در صورتی که کلمه مورد جستجو در دیتابیس وجود داشته باشد لینک های مربوط به آن را به ترتیب امتیاز رتبه به فایل index.balde.php بر می‌گرداند.

Index.blade.php: این فایل ظاهر موتور جستجوگر را نشان می‌دهد. با بهره گیری از فرم ها در html ورودی فیلد جستجو را به کلاس SearchControllerپاس می دهد و پس از پردازش توسط این کلاس لیست لینک های نتیجه را از این کلاس دریافت می‌کند و به عنوان results به کاربر نمایش می دهد.

فایل SearchController.php:

**<?php**

**namespace App\Http\Controllers;**

**use App\Classes\Stemmer;**

**use App\Models\Pagerank;**

**use App\Models\TfIdf;**

**use App\Models\Word;**

**use App\Models\WordDoc;**

**use Illuminate\Http\Request;**

**class SearchController extends Controller**

**{**

**public function index(Request $request)**

**{**

**$s = "";**

**$show\_results = false;**

**$results = [];**

**$search\_method = 1;//1:tf\_idf 2:page rank**

**$word\_id = 0;**

**if ($request->search) {**

**$s = $request->search;**

**$s2 = Stemmer::stem($s);**

**try {**

**$word\_id = $this->findWordId($s2)->id;**

**//page rank method**

**$urls = $this->findWordIdUrls($word\_id);**

**$page\_rank\_array = $this->findPageRankUrls($urls);**

**//tf idf**

**$tf\_idf\_array = $this->findTfIdfUrls($word\_id);**

**if ($search\_method == 1) {**

**$results = $tf\_idf\_array;**

**} else {**

**$results = $page\_rank\_array;**

**}**

**$show\_results = true;**

**} catch (\Exception $e) {**

**$show\_results = true;**

**}**

**}**

**return view('index', compact('s', 'show\_results', 'search\_method', 'results'));**

**}**

**public function findWordId($word)**

**{**

**return Word::where("word", "=", $word)->first();**

**}**

**public function findTfIdfUrls($word\_id)**

**{**

**return TfIdf::select("doc\_url\_id", "score")->where("word\_id", "=", $word\_id)->orderBy("score", "DESC")->distinct()->get();**

**}**

**public function findWordIdUrls($word\_id)**

**{**

**return WordDoc::where("word\_id", "=", $word\_id)->get();**

**}**

**public function findPageRankUrls($urls)**

**{**

**$a = [];**

**if (!empty($urls)) {**

**$a = Pagerank::where("doc\_url\_id", "=", $urls[0]->doc\_url\_id);**

**$size = sizeof($urls);**

**for ($i = 1; $i < $size; $i++) {**

**$a = $a->orWhere("doc\_url\_id", "=", $urls[$i]->doc\_url\_id);**

**}**

**return $a->orderBy("pagerank", "DESC")->distinct()->get();**

**}**

**return $a;**

**}**

**}**

فایل index.blade.php :

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<meta http-equiv="X-UA-Compatible" content="ie=edge">**

**<title>WebSearchEngine</title>**

**<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"**

**integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">**

**<!-- Material Design Bootstrap -->**

**<link href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.8.9/css/mdb.min.css" rel="stylesheet">**

**</head>**

**<body style="background-color: #f7f7f7">**

**<form action="{{route('index')}}" method="get">**

**@csrf**

**<h1 style="text-align: center;margin-top:100px;font-size:70px"><span**

**class="badge badge-dark">WEB SEARCH ENGINE</span>**

**</h1>**

**<div class="card" style="margin-right: 20%;margin-left: 20%;margin-top: 40px;">**

**<div class="card-body">**

**<div class="form-group">**

**<input type="text" class="form-control" name="search" id="search"**

**placeholder="Enter your search query..." value="{{$s}}"**

**required>**

**</div>**

**<div style="text-align: center">**

**<button class="btn btn-primary btn-lg active" type="submit">Search</button>**

**</div>**

**</div>**

**</div>**

**</form>**

**@if($show\_results)**

**<nav aria-label="breadcrumb">**

**<ol class="breadcrumb" style="margin-right: 10%;margin-left:10%;margin-top:30px">**

**<li class="breadcrumb-item active" aria-current="page">{{sizeof($results)}} Results</li>**

**</ol>**

**</nav>**

**<div style="margin-bottom: 100px">**

**@php $i=1; @endphp**

**@foreach($results as $result)**

**@php**

**$url=\App\Models\Url::find($result->doc\_url\_id);**

**$url\_link=$url->url;**

**$url\_title=$url->title;**

**@endphp**

**<a href="{{$url\_link}}">**

**<div class="card" style="margin-right: 10%;margin-left: 10%;margin-top: 30px;padding: 20px">**

**{{$i}}**

**<hr>**

**{{$url\_link}}**

**<hr>**

**{{$url\_title}}**

**</div>**

**</a>**

**@php $i=$i+1; @endphp**

**@endforeach**

**</div>**

**@endif**

**<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"**

**integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"**

**crossorigin="anonymous"></script>**

**<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"**

**integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"**

**crossorigin="anonymous"></script>**

**<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"**

**integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"**

**crossorigin="anonymous"></script>**

**<script type="text/javascript"**

**src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.8.9/js/mdb.min.js"></script>**

**</body>**

**</html>**

**بخش پایگاه داده**

جدول های دیتابیس مطابق با نمودار زیر می‌باشد.

جدول websites وب سایت هایی که عملیات crawling در موردشان انجام می شود را لیست کرده است. در جدول urls لینک های یافت شده در وب سایت ها لیست می شود. جدول words کلمات کلیدی یافت شده در هر url را لیست کرده است. در جدول word\_doc کلمات کلیدی با ذکر url و تعداد تکرار آن آورده شده است. در جدول links ارتباط پدر و فرزندی لینک ها مشخص شده است. در جدول های pagerank و tfidf امتیاز هر یک از url ها ثبت شده است.

