Введение в информационный поиск

Лабораторная работа №2

Списки словопозиций

Выполнила: студентка группы
2371 Танькова Наталья Юрьевна

Преподаватель: Зинченко А.С.

13 октября 2020 г.

1 Ответы на вопросы

1. До внесения изменений:
   (a) В коллекции содержится 678606 термов.
   (b) Размер словаря равен 25964.
2. После добавления обработки стоп-слов:
   (a) В коллекции содержится 673104 терма.
   (b) Размер словаря равен 25812.
3. После добавления обработки термов с помощью алгоритма стемминга Портера:
   (a) В коллекции содержится 463022 терма.
   (b) Размер словаря равен 17419.

2 Результаты запросов

- Результаты запроса "goal"
  - "Antony and Cleopatra Entire Play.htm"
  - "King Lear Entire Play.htm"
  - "Pericles Entire Play.htm"
  - "Winter's Tale Entire Play.htm"
- Результаты запроса "sun"
  - "Titus Andronicus Entire Play.htm"
  - "Troiles and Cressida Entire Play.htm"
  - "Twelfth Night Entire Play.htm"
  - "Two Gentlemen of Verona Entire Play.htm"
  - "Winter's Tale Entire Play.htm"
- Результаты запроса "road"
  - "Merchant of Venice Entire Play.htm"
  - "Much Ado About Nothing Entire Play.htm"
  - "Pericles Entire Play.htm"
  - "Taming of the Shrew Entire Play.htm"
  - "Two Gentlemen of Verona Entire Play.htm"
  - "Winter's Tale Entire Play.htm"
- Результаты запроса стоп-слова "about"
- Результаты запроса "goal and sun"
  - "Winter's Tale Entire Play.htm
- Результаты запроса "sun and about"
- Результаты запроса "sun and road and goal"
  - "Winter's Tale Entire Play.htm"

- Результаты запроса "fled and thrown and leaden and twain"
  - "Othello Entire Play.htm"

3 Программный код

```
[language=Java, caption={Индексация коллекции документов},label=invertedindex]
package invertedindex;

import java.io.*;
import java.util.*;
import org.jsoup.*;
import org.jsoup.nodes.Document;

class Pair {

    int termFrequency;
    LinkedList<Integer> list;

    public Pair(int docID) {
        termFrequency = 1;
        list = new LinkedList<Integer>();
        list.add(docID);
    }

    void addDocument(int docID) {
        this.termFrequency++;
        if (this.list.getLast() != docID) {
            this.list.add(docID);
        }
    }

    void print() {
        for (Iterator I = this.list.iterator(); I.hasNext();) {
            System.out.print(I.next() + " ");
        }
        System.out.println("");
    }
}

public class InvertedIndex {
```

```
List<String> documents = new ArrayList<String>();
Map<String, Pair> index = new HashMap<>();
int countTokens = 0;
LinkedList<String> stopWords;

public InvertedIndex(String path) throws FileNotFoundException {
    stopWords = new LinkedList();
    File file = new File(path);
    Scanner in = new Scanner(file);
    while (in.hasNext()) {
        stopWords.add(in.next());
    }
}

public void indexDocument(String path) throws IOException {
    String extension = "";
    if (path.lastIndexOf(".") > 0) {
        extension = path.substring(path.lastIndexOf(".") + 1);
    }
    if (extension.equals("txt") || extension.equals("doc")
            || extension.equals("docx")) {
        this.indexTxt(path);
    } else if (extension.equals("htm") || extension.equals("html")) {
        this.indexHtml(path);
    } else if (path.contains("http") || path.contains("https")) {
        this.indexUrl(path);
    }
}

public void indexTxt(String path) throws FileNotFoundException {
    File file = new File(path);
    if (!documents.contains(path)) {
        Integer docID = documents.size();
        documents.add(docID, path);
        String line;
        Scanner in = new Scanner(file);
        while (in.hasNextLine()) {
            line = in.nextLine();
```

```
            line = line.toLowerCase();
            String words[] = line.split("[^a-zA-Z0-9_']+");
            this.index(words, docID);
        }
        countTokens += index.size();
        System.out.printf("| %2d | %60s | %5d |%n", docID,
                file.getPath(), index.size());
    }
}

public void indexHtml(String path) throws IOException {
    File htmlFile = new File(path);
    if (!documents.contains(path)) {
        Integer docID = documents.size();
        documents.add(docID, path);
        Document doc = Jsoup.parse(htmlFile, "UTF-8");
        String content = doc.body().text().toLowerCase();
        String[] words = content.split("[^a-zA-Z0-9_']+");
        this.index(words, docID);
        countTokens += index.size();
        System.out.printf("| %2d | %60s | %5d |%n", docID,
                htmlFile.getPath(), index.size());
    }
}

public void indexUrl(String path) throws IOException {
    if (!documents.contains(path)) {
        Integer docID = documents.size();
        documents.add(docID, path);
        Document doc = Jsoup.connect(path).get();
        String content = doc.body().text().toLowerCase();
        String[] words = content.split("[^a-zA-Z0-9_']+");
        this.index(words, docID);
        countTokens += index.size();
        System.out.printf("| %2d | %60s | %5d |%n", docID, path,
                index.size());
    }
}
```

```
public void index(String[] words, Integer docID) {
    boolean isExist = false;
    for (int i = 0; i < words.length; i++) {
        Pair idx = index.get(words[i]);
        Stemmer stemmer = new Stemmer();
        stemmer.add(words[i].toCharArray(), words[i].length());
        stemmer.stem();
        String term = stemmer.toString();
        isExist = false;
        if (idx == null) {
            idx = new Pair(docID);
            if (!this.stopWords.contains(term)) {
                index.put(term, idx);
                isExist = true;
            }
        }
        if (!this.stopWords.contains(term) && (isExist == true
                || idx.list.getLast() != docID)) {
            idx.addDocument(docID);
        }
    }
}

public void indexCollection(String folder) throws IOException {
    File dir = new File(folder);
    String[] files = dir.list();
    for (int i = 0; i < files.length; i++) {
        this.indexDocument(folder + "\\" + files[i]);
    }
}

public static LinkedList<Integer> getIntersection(
        LinkedList<Integer> list1, LinkedList<Integer> list2) {
    LinkedList<Integer> intersection = new LinkedList();
    Iterator<Integer> i = list1.iterator(), j = list2.iterator();
    int element1 = i.next();
    int element2 = j.next();
```

```
    int k = 2;
    int size = list1.size() + list2.size();
    do {
        if (element1 == element2) {
            if (intersection.isEmpty()) {
                intersection.add(element1);
            } else if (intersection.getLast() != element1) {
                intersection.add(element1);
            }
            if (i.hasNext()) {
                element1 = i.next();
            }
            if (j.hasNext()) {
                element2 = j.next();
            }

        }
        if (element1 < element2 && i.hasNext()) {
            element1 = i.next();
        }
        if (element2 < element1 && j.hasNext()) {
            element2 = j.next();
        }
        k++;
    } while (k <= size);
    return intersection;
}

public List<Integer> executeQuery(String query) {
    query = query.toLowerCase();
    String s[] = query.split(" and ");
    LinkedList<Integer> documents = new LinkedList();
    if (s.length < 2) {
        if (this.index.get(query) != null) {
            documents = this.index.get(query).list;
        }
    } else if (this.index.get(s[0]) != null
            && this.index.get(s[1]) != null) {
```

```
            int i = 2;
            documents = getIntersection(this.index.get(s[0]).list,
                    this.index.get(s[1]).list);
            while (i < s.length && documents != null
                    && this.index.get(s[i]) != null) {
                documents = getIntersection(documents,
                        this.index.get(s[i]).list);
                i++;
            }
        }
        return documents;
    }

    public void printDocuments(List query) {
        Integer docID;
        if (query != null) {
            for (Iterator i = query.iterator(); i.hasNext();) {
                docID = (Integer) i.next();
                String s = this.documents.get(docID);
                String[] s1 = s.split("\\\\");
                System.out.println(s1[1]);
            }
        }
    }

    public static void main(String[] args) throws IOException {
        InvertedIndex myIndex = new InvertedIndex("stop_words.txt");
        String path = "collection_html";
        myIndex.indexCollection(path);
        String query1 = "goal";
        String query2 = "sun";
        String query3 = "road";
        String query4 = "about";
        String query5 = "goal and sun";
        String query6 = "sun and about";
        String query7 = "sun and road and goal";
        String query8 = "fled and thrown and leaden and twain";
```

```
        System.out.println("");
        System.out.println(query1);
        myIndex.printDocuments(myIndex.executeQuery(query1));
        System.out.println("");
        System.out.println(query2);
        myIndex.printDocuments(myIndex.executeQuery(query2));
        System.out.println("");
        System.out.println(query3);
        myIndex.printDocuments(myIndex.executeQuery(query3));
        System.out.println("");
        System.out.println(query4);
        myIndex.printDocuments(myIndex.executeQuery(query4));
        System.out.println("");
        System.out.println(query5);
        myIndex.printDocuments(myIndex.executeQuery(query5));
        System.out.println("");
        System.out.println(query6);
        myIndex.printDocuments(myIndex.executeQuery(query6));
        System.out.println("");
        System.out.println(query7);
        myIndex.printDocuments(myIndex.executeQuery(query7));
        System.out.println("");
        System.out.println(query8);
        myIndex.printDocuments(myIndex.executeQuery(query8));
        System.out.println("");
        System.out.println("countTokens = " + myIndex.countTokens);
        System.out.println("index.size = " + myIndex.index.size());
    }
}
```