Übung 01

Arbeitsaufwand insgesamt: 6h

Inhaltsverzeichnis

Teil 1 – Mustersuchalgorithmen

Ziel dieser Übung ist die Implementierung der bereits im 2. Semester besprochenen Mustersuchalgorithmen BruteSearch, BoyerMooreSearch und KnuthMorisPrathSearch.

Lösungsidee:

Als erstes würde ich die Suchalgorithmen in gleichnamigen Klassen implementieren.

Die Algorithmen sind schnell zusammengefasst:

- BruteSearch ist ineffizient und verschiebt im Fehlerfall immer nur um ein Zeichen.
- BoyerMooreSearch verwendet eine zuvor erstellte, Muster-Spezifische Offset-Tabelle und bestimmt daraus im Fehlerfall, um wie viele Stellen verschoben werden darf. Außerdem wird das Muster von hinten nach vorne verglichen.
    - Eignet sich vor allem für lange Muster.
- KnuthMorisPrathSearch basiert auf einer Sprungtabelle, welche aufgrund der Präfixe, welche gleichzeitig auch Suffixe des bereits gematchten Musters sind, die Sprungweite im Fehlerfall bestimmt.
    - Eignet sich vor allem für kleine Alphabete und Muster mit sich wiederholenden Teilen.

Zusätzlich dazu, würde ich die Methode „search“ sowie die Hilfsmethoden überladen, sodass ein DataCollector mitgegeben werden kann. Der DataCollector enthält einfache Zählvariablen für jeden Operationstyp (bzw. eine Map, um die Buchstaben zu zählen) und zugehörige Methoden, die es einfach machen diese Werte zu inkrementieren bzw. um einen bestimmten Wert zu erhöhen. Zusätzlich dazu müssen in der abgeänderten „search“ Methode auch nach jeder Operation (oder zumindest gesammelt, nach einem Block) die jeweiligen Methoden zum Zählen der Operationen aufgerufen werden.

Lösung:

Als IDEA Projekt im Archiv
Nachfolgender Source-Code

Source-Code:

SearchTest.java

```
package swp4.ue01.part1.test;

import swp4.ue01.part1.DataCollection.DataCollector;
import swp4.ue01.part1.search.BMSearch;
import swp4.ue01.part1.search.BruteSearch;
import swp4.ue01.part1.search.KMPSearch;

import java.util.Random;

public class SearchTest {
    public static void main(String[] args) {
        normalTest();
        //emptyTest();
        //binaryTest();
        //baseTest();
        //aminoTest();
        //asciiTest();
    }

    static String getRandomText(int length, int chars) {
        String text="";
        Random r = new Random();
        for(int i=0; i < length; i++)
        {
            text+=(char)r.nextInt(chars);
        }
        return text;
    }

    static void normalTest() {
        int pos1 = BruteSearch.search("Das ist ein Text!", "Text");
        System.out.println("Pattern found at position "+pos1+".");

        int pos2 = BMSearch.search("Das ist ein Text!", "Text");
        System.out.println("Pattern found at position "+pos2+".");

        int pos3 = KMPSearch.search("Das ist ein Text!", "Text");
        System.out.println("Pattern found at position "+pos3+".");
    }
    static void emptyTest() {
        int pos1 = BruteSearch.search("Das ist ein Text!", "");
        System.out.println("Pattern found at position "+pos1+".");

        int pos2 = BMSearch.search("", "Text");
        System.out.println("Pattern found at position "+pos2+".");

        int pos3 = KMPSearch.search("", "");
        System.out.println("Pattern found at position "+pos3+".");
    }

    static void binaryTest() {
        DataCollector dc1 = new DataCollector();
        DataCollector dc2 = new DataCollector();
        DataCollector dc3 = new DataCollector();
```

```
        for(int len=10; len <= 1000; len*=10) {
            for(int i=0; i < 1000; i++) {
                String text = getRandomText(len, 2);
                String pattern = getRandomText(10,2);

                int pos1 = BruteSearch.search(text, pattern, 0, dc1);
                int pos2 = BruteSearch.search(text, pattern, 0, dc1);
                int pos3 = BruteSearch.search(text, pattern, 0, dc1);
            }
            dc1.writeToFile("datacollector/BF_binary_"+len+".csv");
            dc2.writeToFile("datacollector/BM_binary_"+len+".csv");
            dc3.writeToFile("datacollector/KMP_binary_"+len+".csv");
        }


    }
    static void baseTest() {
        DataCollector dc1 = new DataCollector();
        DataCollector dc2 = new DataCollector();
        DataCollector dc3 = new DataCollector();

        for(int len=10; len <= 1000; len*=10) {
            for(int i=0; i < 1000; i++) {
                String text = getRandomText(len, 2);
                String pattern = getRandomText(10,2);

                int pos1 = BruteSearch.search(text, pattern, 0, dc1);
                int pos2 = BruteSearch.search(text, pattern, 0, dc2);
                int pos3 = BruteSearch.search(text, pattern, 0, dc3);
            }
            dc1.writeToFile("datacollector/BF_basepair_"+len+".csv");
            dc2.writeToFile("datacollector/BM_basepair_"+len+".csv");
            dc3.writeToFile("datacollector/KMP_basepair_"+len+".csv");
        }

    }
    static void aminoTest() {
        DataCollector dc1 = new DataCollector();
        DataCollector dc2 = new DataCollector();
        DataCollector dc3 = new DataCollector();

        for(int len=10; len <= 1000; len*=10) {
            for(int i=0; i < 1000; i++) {
                String text = getRandomText(len, 2);
                String pattern = getRandomText(10,2);

                int pos1 = BruteSearch.search(text, pattern, 0, dc1);
                int pos2 = BruteSearch.search(text, pattern, 0, dc2);
                int pos3 = BruteSearch.search(text, pattern, 0, dc3);
            }
            dc1.writeToFile("datacollector/BF_amino_"+len+".csv");
            dc2.writeToFile("datacollector/BM_amino_"+len+".csv");
            dc3.writeToFile("datacollector/KMP_amino_"+len+".csv");
        }

    }
    static void asciiTest() {
        DataCollector dc1 = new DataCollector();
```

```
        DataCollector dc2 = new DataCollector();
        DataCollector dc3 = new DataCollector();

        for(int len=10; len <= 1000; len*=10) {
            for(int i=0; i < 1000; i++) {
                String text = getRandomText(len, 2);
                String pattern = getRandomText(10,2);

                int pos1 = BruteSearch.search(text, pattern, 0, dc1);
                int pos2 = BruteSearch.search(text, pattern, 0, dc2);
                int pos3 = BruteSearch.search(text, pattern, 0, dc3);
            }
            dc1.writeToFile("datacollector/BF_ascii_"+len+".csv");
            dc2.writeToFile("datacollector/BM_ascii_"+len+".csv");
            dc3.writeToFile("datacollector/KMP_ascii_"+len+".csv");
        }

    }

}
```

BruteSearch.java

```
package swp4.ue01.part1.search;

import swp4.ue01.part1.DataCollection.DataCollector;

public class BruteSearch {

    public static int search(String text, String pattern) {
        return search(text,pattern,0);
    }
    public static int search(String text, String pattern, int offset) {
        if(pattern.isEmpty() || text.isEmpty())
        {
            return -1;
        }

        int foundPos = -1;
        int pos = offset;
        int tLen = text.length();
        int pLen = pattern.length();

        while(pos>=0 && pos<tLen-pLen+1 && foundPos == -1)
        {
            int pIndex = 0;

            while(pIndex<pLen &&
pattern.charAt(pIndex)==text.charAt(pos+pIndex)) {
                pIndex++;
            }

            if(pIndex==pLen)
            {
                foundPos = pos;
            }

            pos++;
        }

        return foundPos;
    }
    public static int search(String text, String pattern, int offset,
DataCollector dc) {
        if(pattern.isEmpty() || text.isEmpty())
        {
            return -1;
        }
        dc.logAssOp(3); // parameters, excluding dc

        int foundPos = -1;
        int pos = offset;
        int tLen = text.length();
        int pLen = pattern.length();
        dc.logAssOp(4); // above parameters

        // log operations for exit statement (not executed within while)
        dc.logLogOp(4);
        dc.logAddOp(2);
        while(pos>=0 && pos<tLen-pLen+1 && foundPos == -1)
```

```
        {
            dc.logLogOp(4);
            dc.logAddOp(2);

            int pIndex = 0;
            dc.logAssOp();

            // log operations for exit of while statement (not executed within
while)
            dc.logLogOp(3);
            dc.logIndOp(2);
            dc.logAddOp();
            while(pIndex<pLen &&
pattern.charAt(pIndex)==text.charAt(pos+pIndex)) {
                dc.logCharCmp(text.charAt(pos+pIndex),true);
                pIndex++;
                // while statement
                dc.logLogOp(3);
                dc.logIndOp(2);
                dc.logAddOp(2); // once in while statement, once for pIndex
                dc.logAssOp();
            }

            dc.logCmp(); // if
            if(pIndex==pLen)
            {
                foundPos = pos;
                dc.logAssOp(); // foundPos
            }

            if(pIndex < pattern.length()) {
                // not counted in dc, due to the fact that this function
caused the need for the if statement
                dc.logCharCmp(text.charAt(pos+pIndex),false);
            }


            dc.logAddOp(); // pos
            dc.logAssOp();
            pos++;
        }


        dc.logAssOp(); // return value
        return foundPos;
    }
}
```

BMSearch.java

```
package swp4.ue01.part1.search;

import swp4.ue01.part1.DataCollection.DataCollector;
import java.util.HashMap;
import java.util.Map;


public class BMSearch {

    public static int search(String text, String pattern) {
        return search(text, pattern, 0);
    }
    public static int search(String text, String pattern, int offset) {
        Map<Character, Integer> table = makeOffsetTable(pattern);

        int tLen = text.length();
        int pLen = pattern.length();
        int pos = pLen + offset - 1;
        int foundPos = -1;

        while (pos >= pLen - 1 && pos < tLen && foundPos == -1) {
            int pIndex = 0;

            while (pIndex < pLen && pattern.charAt(pLen - pIndex - 1) ==
text.charAt(pos - pIndex)) {
                pIndex++;
            }

            if (pIndex == pLen) {
                foundPos = pos - (pLen - 1);
            } else {
                if (table.containsKey(text.charAt(pos - pIndex))) {
                    pos += table.get(text.charAt(pos - pIndex));
                } else {
                    pos += pLen;
                }
            }
        }

        return foundPos;
    }
    public static int search(String text, String pattern, int offset,
DataCollector dc) {
        Map<Character, Integer> table = makeOffsetTable(pattern, dc);
        dc.logAssOp(4); // assign table, and also pass the parameters,
excluding dc

        int tLen = text.length();
        int pLen = pattern.length();
        int pos = pLen + offset - 1;
        int foundPos = -1;
        dc.logAssOp(4); // above parameters
        dc.logAddOp();


         // While (Exit)
        dc.logLogOp(5);
```

```
        dc.logAddOp();
        while (pos >= pLen - 1 && pos < tLen && foundPos == -1) {
            // while
            dc.logLogOp(5);
            dc.logAddOp();

            int pIndex = 0;
            dc.logAssOp(); // pIndex

            //while
            dc.logLogOp(3);
            dc.logAddOp(3);
            dc.logIndOp(2);
            while (pIndex < pLen && pattern.charAt(pLen - pIndex - 1) ==
text.charAt(pos - pIndex)) {
                //while
                dc.logCharCmp(text.charAt(pos - pIndex),true);
                dc.logLogOp(3);
                dc.logAddOp(3);
                dc.logIndOp(2);
                pIndex++;
                dc.logAddOp();
                dc.logAssOp();
            }

            // while comparison
            if(pIndex < pLen) {
                dc.logCharCmp(text.charAt(pos - pIndex),false);
            }

            dc.logLogOp(); //if
            if (pIndex == pLen) {
                foundPos = pos - (pLen - 1);
                dc.logAssOp(); // foundPos
                dc.logAddOp(2);
            } else {
                dc.logLogOp(); // bloew if
                dc.logAddOp();
                dc.logIndOp();
                if (table.containsKey(text.charAt(pos - pIndex))) {
                    pos += table.get(text.charAt(pos - pIndex));
                    dc.logIndOp(); // pos
                    dc.logAddOp(2);
                    dc.logAssOp();
                } else {
                    pos += pLen; // pos
                    dc.logAddOp();
                    dc.logAssOp();
                }
            }
            dc.logCmp(); // comparison loop
        }

        dc.logAssOp(); // return
        return foundPos;
    }

    private static Map<Character,Integer> makeOffsetTable(String pattern) {
```

```
        Map<Character,Integer> table = new HashMap<Character,Integer>();

        for(int i=0; i < pattern.length()-1; i++) {
            table.put(pattern.charAt(i),pattern.length()-1-i);
        }

        return table;
    }
    private static Map<Character,Integer> makeOffsetTable(String pattern,
DataCollector dc) {
        Map<Character,Integer> table = new HashMap<Character,Integer>();
        dc.logAssOp();

        dc.logAssOp();
        dc.logLogOp();
        dc.logAddOp();
        for(int i=0; i < pattern.length()-1; i++) {
            dc.logLogOp();
            dc.logAddOp();

            table.put(pattern.charAt(i),pattern.length()-1-i);
            dc.logAssOp();
            dc.logIndOp();
            dc.logAddOp(2);

            dc.logAddOp();
        }

        return table;
    }

}
```

KMPSearch.java

```
package swp4.ue01.part1.search;

import swp4.ue01.part1.DataCollection.DataCollector;

public class KMPSearch {


    public static int search(String text, String pattern, int offset,
DataCollector dc) {
        dc.logAssOp(3); // parameters, excluding dc

        if(pattern.isEmpty() || text.isEmpty())
        {
            return -1;
        }

        int[] lp = getLeapTbl(pattern, dc);
        int pos = offset;
        int pIndex = 0;
        dc.logAssOp(3); // variables above

        dc.logLogOp(3); // while (exit)
        while(pos < text.length() && pIndex < pattern.length()) {
            dc.logLogOp(3); // while

            dc.logLogOp(2); // while (/exit)
            dc.logIndOp(2);
            while(pIndex >= 0 && text.charAt(pos) != pattern.charAt(pIndex)) {
                dc.logLogOp(2); // while
                dc.logIndOp(2);
                dc.logCharCmp(text.charAt(pos),true);
                pIndex = lp[pIndex];
                dc.logAssOp();
                dc.logIndOp();
            }

            if(pIndex < pattern.length()) {
                dc.logCharCmp(text.charAt(pos),false);
            }

            pos++;
            pIndex++;
            dc.logAssOp(2);
            dc.logAddOp(2);

            dc.logCmp();
        }

        dc.logLogOp();
        dc.logAddOp(1);
        dc.logAssOp();

        return pIndex==pattern.length() ? pos - pattern.length() : -1;
    }
    public static int search(String text, String pattern, int offset) {
        if(pattern.isEmpty() || text.isEmpty())
        {
```

```
            return -1;
        }

        int[] lp = getLeapTbl(pattern);
        int pos_text = offset;
        int pos_pattern = 0;

        while(pos_text < text.length() && pos_pattern < pattern.length()) {
            while(pos_pattern >= 0 && text.charAt(pos_text) !=
pattern.charAt(pos_pattern)) {
                pos_pattern = lp[pos_pattern];
            }

            pos_text++;
            pos_pattern++;
        }

        return pos_pattern==pattern.length() ? pos_text - pattern.length() : -
1;
    }
    public static int search(String text, String pattern){
        return search(text,pattern,0);
    }


    static int[] getLeapTbl(String pattern) {
        if(pattern.isEmpty()) {
            return null;
        }

        int[] leapTbl = new int[pattern.length()];
        int pIndex = 0;
        int prefixSize = -1;

        leapTbl[0] = prefixSize;
        while(pIndex < pattern.length()) {
            while(prefixSize >= 0 && pattern.charAt(pIndex) !=
pattern.charAt(prefixSize)) {
                prefixSize = leapTbl[prefixSize];
            }

            pIndex++;
            prefixSize++;
            if(pIndex < pattern.length()) {
                leapTbl[pIndex] = prefixSize;
            }
        }

        return leapTbl;
    }
    static int[] getLeapTbl(String pattern, DataCollector dc) {
        if(pattern.isEmpty()) {
            return null;
        }

        dc.logAssOp(2); // parameters

        int[] leapTbl = new int[pattern.length()];
```

```
        int pIndex = 0;
        int prefixSize = -1;
        dc.logAssOp(3);

        leapTbl[0] = prefixSize;
        dc.logIndOp();
        dc.logAssOp();

        dc.logLogOp();
        while(pIndex < pattern.length()) {
            dc.logLogOp();

            dc.logLogOp(2);
            dc.logIndOp(2);
            while(prefixSize >= 0 && pattern.charAt(pIndex) !=
pattern.charAt(prefixSize)) {
                dc.logLogOp(2);
                dc.logIndOp(2);

                prefixSize = leapTbl[prefixSize];
                dc.logAssOp();
                dc.logIndOp();
            }

            pIndex++;
            prefixSize++;
            dc.logAssOp(2);
            dc.logAddOp(2);

            dc.logLogOp();
            if(pIndex < pattern.length()) {
                leapTbl[pIndex] = prefixSize;
                dc.logIndOp();
                dc.logAssOp();
            }
        }

        dc.logAssOp();
        return leapTbl;
    }
}
```

CharacterStat.java

```java
package swp4.ue01.part1.DataCollection;

public class CharacterStat {
    private int totalComparisons;
    private int successfulComparisons;

    public CharacterStat() {
        totalComparisons = 0;
        successfulComparisons = 0;
    }

    public void updateStats(boolean success) {
        totalComparisons++;
        if(success) {
            successfulComparisons++;
        }
    }

    public int getSuccessfulComparisons() {
        return successfulComparisons;
    }

    public int getFailedComparisons() {
        return totalComparisons - successfulComparisons;
    }


}
```

DataCollector.java

```java
package swp4.ue01.part1.DataCollection;

import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class DataCollector {
    int charCmp;
    int logOp;
    int addOp;
    int multOp;
    int indOp;
    int cmp;
    int assOp;
    Map<Character,CharacterStat> charCmpTable;

    public DataCollector() {
        charCmp = 0;
        logOp = 0;
        addOp = 0;
        multOp = 0;
        indOp = 0;
        cmp = 0;
        assOp = 0;
        charCmpTable = new HashMap<>();
    }

    public void clear() {
        charCmp = 0;
        logOp = 0;
        addOp = 0;
        multOp = 0;
        indOp = 0;
        cmp = 0;
        assOp = 0;
        charCmpTable.clear();
    }

    public void logCharCmp(char letter, boolean success) {
        charCmp++;
        CharacterStat stat = charCmpTable.get(letter);
        if(stat != null) {
            stat.updateStats(success);
        } else {
            charCmpTable.put(letter, new CharacterStat());
            charCmpTable.get(letter).updateStats(success);
        }
    }
    public void logLogOp() {
        logOp++;
    }
    public void logAddOp() {
        addOp++;
    }
    public void logMultOp() {
        multOp++;
```

```
    }
    public void logIndOp() {
        indOp++;
    }
    public void logCmp() {
        cmp++;
    }
    public void logAssOp() {
        assOp++;
    }

    public void logLogOp(int amount) {
        logOp+=amount;
    }
    public void logAddOp(int amount) {
        addOp+=amount;
    }
    public void logMultOp(int amount) {
        multOp+=amount;
    }
    public void logIndOp(int amount) {
        indOp+=amount;
    }
    public void logCmp(int amount) {
        cmp+=amount;
    }
    public void logAssOp(int amount) {
        assOp+=amount;
    }

    public void printStatistics() {
        System.out.println("Statistics of comparison:");
        System.out.println("Chars compared:\t\t\t\t"+charCmp);
        System.out.println("Logical operations:\t\t\t"+logOp);
        System.out.println("Additive operations:\t\t\t"+addOp);
        System.out.println("Multiplicative operations:\t"+multOp);
        System.out.println("Index operations:\t\t\t"+indOp);
        System.out.println("Attempted comparisons:\t\t"+cmp);
        System.out.println("Assignment operations:\t\t"+assOp);
    }

    public void writeToFile(String filename) {
        try(FileWriter fout = new FileWriter(filename)) {
            fout.write("Statistics of comparison:\n");
            fout.write("Characters compared;Logical operations;Additive
operations;Multiplicative operations;Index operations;Attempted
comparisons;Assignment operations;\n");

fout.write(charCmp+";"+logOp+";"+addOp+";"+multOp+";"+indOp+";"+cmp+";"+assOp+
"\n");
            fout.write("\n");

            String successLine = "";
            String failLine = "";
            for(char key : charCmpTable.keySet()) {
                fout.write(key+";");
                successLine +=
charCmpTable.get(key).getSuccessfulComparisons()+";";
```

```
                failLine += charCmpTable.get(key).getFailedComparisons()+";";
            }
            fout.write("\n");
            fout.write(successLine+"\n");
            fout.write(failLine+"\n");

        } catch(IOException e)
        {
            System.out.println("Coult not write to file: " + filename);
            e.printStackTrace();
        }
    }
}
```

Testfälle

- Normale Benützung
- Leere Eingaben für Muster/Text/Beides
- Datacollector CSVs

Normale Benützung:

```
Pattern found at position 12.
Pattern found at position 12.
Pattern found at position 12.
```

Leere Eingaben:

```
Pattern found at position -1.
Pattern found at position -1.
Pattern found at position -1.
```

Die Ausgaben für die jeweiligen Alphabete, Längen und Algorithmen ist im Ordner „datacollector“ im Projektordner zu finden.

Teil 2 – Bacon-Index

Anschließend zu Beispiel 1 soll noch das Wissen aus der Übung eingesetzt werden, um einen Graph aus Schauspielern/Filmen aufzubauen und mithilfe von diesem den sogenannten Bacon-Index zu berechnen. Der Kevin-Bacon Index sagt aus, über „wie viele Filme“ die Verbindung dieser Schauspieler verläuft.

Lösungsidee:

Ursprünglich wollte ich dieses Beispiel mithilfe einer Adjazenzmatrix lösen (Dreiecksmatrix mit ArrayLists), jedoch war die Datenmenge zu groß für den Heap. Im Retrospekt macht eine Adjazenzliste auch mehr Sinn, da das Verhältnis von Knoten zu Kanten zu gering ist (viele „unnütze“ „false“ Werte in der Matrix).

Bei der Berechnung des Bacon-Index wird mit einem der beiden Akteure gestartet und von diesem aus alle Nachbarn bestimmt (= genau EIN Film Distanz). Dieser Prozess wird dann für die erhaltenen Akteure wiederholt, bis sich die Zielperson unter den Akteuren befindet oder alle Knoten durchsucht wurden.

Um Performanz zu verbessern (und um vertretbare Suchzeiten zu erhalten) musste ich zusätzlich mehreren Caching-Algorithmen mittels Maps und ArrayLists arbeiten. Ein Beispiel dafür wäre, die Bereiche der Filme (Start- und Endindex in der Adjazenzliste) festzuhalten, damit später bei der Suche nach Film-Zuweisungen nur bedeutend geringere Bereiche durchsucht werden müssen.

Lösung:

Als Java Projekt im Archiv
Nachfolgender Source-Code

Source-Code:

MovieTest.java

```
package swp4.ue01.part2.test;

import swp4.ue01.part2.collections.graph.MovieGraph;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class MovieTest {
    public static void main(String[] args) {
        MovieGraph m = new MovieGraph("movies.txt");

        //m.testRandomBacons();

        String name1 = " ";
        String name2 = " ";
        try {

            System.out.println("Press enter without content to exit.");
            // loop as long as
            while(!name1.isEmpty() && !name2.isEmpty()) {
                System.out.println("Please enter the first actors name:");
                name1 = readInputFromSystemIn();

                if(!name1.isEmpty()) {
                    System.out.println("Please enter the second actors
name:");
                    name2 = readInputFromSystemIn();
                }

                if(!name1.isEmpty() && !name2.isEmpty()) {
                    System.out.println("Their bacon-index is:
"+m.getBaconIndex(name1,name2));
                } else {
                    System.out.println("Enter detected. Exiting...");
                }
                System.out.println();
            }
        } catch(Exception e) {
            System.err.println("Couldn't read from console.");
            e.printStackTrace();
        }

    }

    public static String readInputFromSystemIn() throws Exception {
        BufferedReader reader = new BufferedReader( new
InputStreamReader( System.in ) );
        return reader.readLine();
    }

}
```

Graph.java

```java
package swp4.ue01.part2.collections.graph;

public interface Graph {

    boolean addEdge( int src, int dest );

    boolean removeEdge( int src, int dest );

    boolean hasEdge( int src, int dest );

}
```

MovieGraph.java

```
package swp4.ue01.part2.collections.graph;

import swp4.ue01.part2.calculation.BaconIndexCalculator;
import swp4.ue01.part2.collections.MovieSymbolTable;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Map.Entry;
import java.util.Random;

public class MovieGraph implements Graph {



    private MovieSymbolTable mst;
    private ArrayList<Entry<Integer,Integer>> adjacencyList;
    private ArrayList<Integer> movieStartIdx;


    public MovieGraph(String filename) {
        this.mst = new MovieSymbolTable();
        this.adjacencyList = new ArrayList<>();
        this.movieStartIdx = new ArrayList<>();

        // open file
        try (BufferedReader bReader = new BufferedReader(new
InputStreamReader(new FileInputStream(filename)))) {
            String line;
            int entryNum = 0;

            // read every line
            while((line=bReader.readLine()) != null) {
                // cache the start indices of movies in the adjacencylist.
                movieStartIdx.add(entryNum);
                int movieNum = -1;
                // splits the line into movies + actors
                String[] elements = line.split("/");

                // deal with each new item in the line
                for(int i=0; i < elements.length; i++) {
                    String value = elements[i].strip();

                    // if node with specific value doesn't exist, create it.
                    if(mst.getIndex(value) == -1) {
                        if(i == 0) { // if movie
                            mst.add(value, mst.size(), true);
                            movieNum = mst.size()-1;
                        } else {
                            mst.add(value, mst.size(), false);
                        }

                        // if the new node is a person, also add an edge to
the movie (first element in line)
```

```
                        if(i != 0) { // person
                            addEdge(mst.size()-1, movieNum);
                            entryNum++;
                        }
                    } else {
                        // if actors or movies already exist, all thats left
is to create the edges
                        if(i == 0)
                        {
                            // this shouldn't happen, but just in case. (since
every line should be a NEW movie)
                            movieNum = mst.getIndex(value);
                        } else {
                            addEdge(mst.getIndex(value), movieNum);
                            entryNum++;
                        }
                    }
                }
            }
            // finally, add a "cached" entry AFTER all movies are finished.
(to have continuous areas between index and index+1)
            movieStartIdx.add(entryNum);
        } catch(IOException e) {
            System.err.println("Could not read file " + filename);
            e.printStackTrace();
        }
        System.out.println("Successfully created graph for "+filename+".");

        // finally, use the graph/adjacencylist to calculate the bacon index.
    }

    @Override
    public boolean addEdge(int src, int dest) {

        // add the edge, so the movie is always the first node
        if(mst.isMovie(src)) {
            return adjacencyList.add(new AbstractMap.SimpleEntry(src,dest));
        } else {
            return adjacencyList.add(new AbstractMap.SimpleEntry(dest,src));
        }
    }

    @Override
    public boolean removeEdge(int src, int dest) {
        if(mst.isMovie(src)) {
            return adjacencyList.remove(new
AbstractMap.SimpleEntry(src,dest));
        } else {
            return adjacencyList.remove(new
AbstractMap.SimpleEntry(dest,src));
        }
    }

    @Override
    public boolean hasEdge(int src, int dest) {
        if(src == dest) {
            return true;
        }
```

```
        return hasEdge(0,adjacencyList.size());
    }

    // determines whether the nodes have an edge between a specific range
(speeds up the process significantly, by utilizing movie index caching)
    public boolean hasEdge(int src, int dest, int start, int end) {
        if(src == dest) {
            return true;
        }
        // make sure movie is first node...
        if(mst.isMovie(dest)) {
            int tmp = src;
            src = dest;
            dest = tmp;
        }

        // search in the designated range
        for(int i=start; i < end; i++) {
            if(adjacencyList.get(i).getValue() == dest)
            {
                return true;
            }
        }

        return false;
    }

    // returns a list of actors that are connected to the initial actor via
movies
    public ArrayList<Integer> getConnectedActors(int index, ArrayList<Integer>
scanned) {
        ArrayList<Integer> values = new ArrayList();

        // if index a movie, terminate.
        if(mst.isMovie(index))
        {
            System.out.println("Specified index is a movie, not an actor.
Terminating.");
            return values;
        }

        ArrayList<Integer> allMovies = mst.getMovies();
        // Iterate through all movies, and check if the actor starrs in it
        for(int mi=0; mi<allMovies.size(); mi++) {
            int movieIdx = allMovies.get(mi);
            int mstart = movieStartIdx.get(mi);
            int mend = movieStartIdx.get(mi+1);
            if(hasEdge(index,movieIdx,mstart,mend)
&& !scanned.contains(movieIdx)) {
                // Then log all actors that play in that movie to the list
                for(int mci=mstart; mci < mend; mci++) {
                    if(adjacencyList.get(mci).getValue() != index)
                    {
                        values.add(adjacencyList.get(mci).getValue());
                    }
                }
                // also, mark the items as scanned.
                scanned.add(movieIdx);
```

```
                scanned.add(index);
            }

        }

        // finally return the arraylist
        return values;
    }

    // returns the Bacon-Index for the given actors
    public int getBaconIndex(String name1, String name2) {
        return BaconIndexCalculator.calculateBaconIndex(this, this.mst, name1,
name2);
    }

    public void testRandomBacons() {
        // random actors to check if bacon-index seems feasible
        Random rand = new Random();
        for(int i=0; i < 1000; i++) { // Try for 1000 random pairs
            int p1 = 1;//rand.nextInt(mst.size());
            int p2 = 2;//rand.nextInt(mst.size());

            if(mst.isActor(p1) && mst.isActor(p2)) {
                System.out.println("'"+mst.getValue(p1)+"' and
'"+mst.getValue(p2)+"':"+BaconIndexCalculator.calculateBaconIndex(this, mst,
mst.getValue(p1), mst.getValue(p2)));
            }
        }
    }

}
```

MovieNode.java

```java
package swp4.ue01.part2.collections;

public class MovieNode {
    private String value;
    private boolean movie;

    // create a movienode
    public MovieNode(String value, boolean movie) {
        this.value = value;
        this.movie = movie;
    }

    // return the nodes value
    public String getValue() {
        return value;
    }
    // return whether the node is a movie or not
    public boolean isMovie() {
        return movie;
    }
}
```

MovieSymbolTable.java

```java
package swp4.ue01.part2.collections;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class MovieSymbolTable {
    // holds all the nodes and their values
    private ArrayList<MovieNode> nodes;
    // holds all indexes for movies (to map them easily later)
    private ArrayList<Integer> movieIdx;
    // Map to store the names of actors/movies to their indices
    private Map<String,Integer> nodeHash;

    private int nodeCount;

    public MovieSymbolTable() {
        nodes = new ArrayList<>();
        nodeHash = new HashMap<>();
        nodeCount = 0;
        movieIdx = new ArrayList<>();
    }

    // adds a new movie or actor to the table
    public void add(String value, int nodeId, boolean isMovie) {
        nodes.add(new MovieNode(value,isMovie));
        nodeHash.put(value,nodeCount);

        // if its a movie, also log its index in the separate arraylist
        if(isMovie) {
            movieIdx.add(nodeId);
        }
        nodeCount++;
    }

    // returns the value for the node of the given nodeId
    public String getValue(int nodeId) {
        return nodes.get(nodeId).getValue();
    }

    // returns the index for the node of the given value
    public int getIndex(String value) {
        return nodeHash.get(value) != null ? nodeHash.get(value) : -1;
    }

    // returns whether a node is a movie or not
    public boolean isMovie(int nodeId) {
        return nodes.get(nodeId).isMovie();
    }

    // returns whether a node is a actor or not
    public boolean isActor(int nodeId) {
        return !nodes.get(nodeId).isMovie();
    }

    // returns the amount of nodes.
    public int size() {
```

```
        return nodes.size();
    }

    // returns a list of all movie indices
    public ArrayList<Integer> getMovies() {
        return movieIdx;
    }

}
```

BaconIndexCalculator.java

```
package swp4.ue01.part2.calculation;

import swp4.ue01.part2.collections.MovieSymbolTable;
import swp4.ue01.part2.collections.graph.MovieGraph;

import java.util.ArrayList;

public class BaconIndexCalculator {
    // Calculates the BaconIndex between two actors using the given Graph and
MovieSymboltable
    public static int calculateBaconIndex(MovieGraph graph, MovieSymbolTable
mst, String name1, String name2) {
        // if names dont exist or aren't actors, return -1
        if(mst.getIndex(name1)==-1 || mst.getIndex(name2)==-1 ||
mst.isMovie(mst.getIndex(name1)) || mst.isMovie(mst.getIndex(name2))) {
            System.out.println("Couldn't calculate Bacon index, since the
given names couldn't be found!");
            return -1;
        }

        int targetActorIndex = mst.getIndex(name2);

        if(mst.getIndex(name1) == targetActorIndex) {
            return 0;
        }

        // list all nodes (movies AND actors) that have already been scanned,
to avoid cycles (or additional work)
        ArrayList<Integer> scanned = new ArrayList();
        int baconIndex = 0;

        // Arraylist to store all the current actors, from which the scan goes
forward.
        // Start scan from actor with name1
        ArrayList<Integer> actors = new ArrayList();
        actors.add(mst.getIndex(name1));
        do {
            // hold initial size, to avoid looping past the original dataset.
            int initialSize = actors.size();
            // scan from all nodes inside <actors>
            for(int i=0; i < initialSize; i++) {
                int idx = actors.get(i);
                if(mst.isActor(idx)) {
                    // get all actors that starred in the same movies
                    ArrayList<Integer> tmpNbList =
graph.getConnectedActors(idx, scanned);
                    scanned.add(idx);

                    // add the neighbors to the actors arraylist, so they will
be scanned later.
                    tmpNbList.forEach(actors::add);
                }

                // remove actors that have already been scanned
                actors.remove(i);
                initialSize--;
            }
```

```
            // after each iteration increase bacon-index (= amount of movies
distance from initial actor)
            baconIndex++;
        } while(!actors.isEmpty() && !actors.contains(targetActorIndex));

        // if all nodes have been searched (not found) or index has been
determined, return value
        return actors.isEmpty() ? -1 : baconIndex;
    }
}
```

Testfälle

- Eingabe in der Konsole
- Gleiche Person
- Nichtexistente Akteure
- Bacon-Index von Filmen
- Leeres File
- Abbruch
- Zufällige Personen

Eingabe in der Konsole:

```
Successfully created graph for movies.txt.
Press enter without content to exit.
Please enter the first actors name:
Tygner, Robert
Please enter the second actors name:
White, Miranda
Their bacon-index is: 3
```

Gleiche Person:

```
Successfully created graph for movies.txt.
Press enter without content to exit.
Please enter the first actors name:
White, Miranda
Please enter the second actors name:
White, Miranda
This is the same person!
Their bacon-index is: 0
```

Leeres File:

```
Successfully created graph for empty.txt.
Press enter without content to exit.
Please enter the first actors name:
Williams, Robin
Please enter the second actors name:
Cena, John
Couldn't calculate Bacon index, since the given names couldn't be found!
Their bacon-index is: -1
```

Abbruch:

```
Successfully created graph for empty.txt.
Press enter without content to exit.
Please enter the first actors name:

Enter detected. Exiting...
```

Nichtexistente Akteure:

```
Successfully created graph for movies.txt.
Press enter without content to exit.
Please enter the first actors name:
Johannes
Please enter the second actors name:
Merkel
Couldn't calculate Bacon index, since the given names couldn't be found!
Their bacon-index is: -1
```

Bacon-Index von Filmen:

```
Successfully created graph for movies.txt.
Press enter without content to exit.
Please enter the first actors name:
'Breaker' Morant (1980)
Please enter the second actors name:
'Crocodile' Dundee II (1988)
Couldn't calculate Bacon index, since the given names couldn't be found!
Their bacon-index is: -1
```

Zufällige Akteure:

```
'Muska, Vaclav' and 'Kadani, Otis':4
'McClain, Johnathan (II)' and 'Roussel, Francine':3
'Barbour, Joyce' and 'Dong, Pu Ye':5
'von Dohlen, Lenny' and 'McGraw, Kathy':3
'Hewson, Thomas' and 'Carr, Ben (I)':3
'Garcia, Jon (I)' and 'Szanyi, Monica':2
'Taggart, Rachel' and 'Goryl, Dave':4
'Sinatra Jr., Frank' and 'Davis, Cherry (I)':3
'Rodriguez, Lucy' and 'Reiner, Richie':3
'Brimley, Wilford' and 'Davies, Erik':2
'Cromwell, J.T.' and 'Buchette, Oliver':5
'Alvarez, Dale' and 'Carver, Mary':3
'Gómez, Paz' and 'Yankovsky, Filipp':4
'Wilson, Hal (I)' and 'Daquila, Lawrence':3
'Brown, Bryan (I)' and 'Henderson, Dick (II)':1
```