Aufgabe 1: Wörter Aufräumen

Team-ID: 00325

Team-Name: pwned

Bearbeiter/-innen dieser Aufgabe:
Anton Ferdinand Althoff

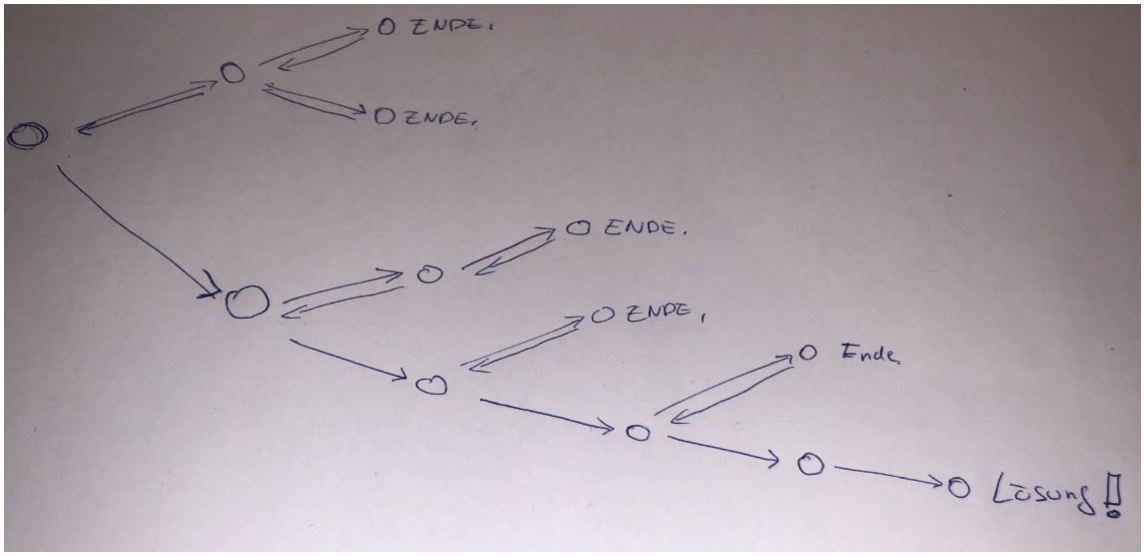
23. Oktober 2020

Inhaltsverzeichnis

Lösungsidee

Ich habe mich entschieden, das Problem rekursiv via Backtracking zu lösen, da der Code sehr übersichtlich ist. Wir haben einen Lückentext, wobei in manche Lücken mehrere Wörter reinpassen, z.B. in „_i_“ passt „die“, aber auch „sie“. Wenn wir zuerst ein Wort einsetzen, kann es uns später fehlen. Daher können wir nicht einfach die Liste der Wörter durchgehen und das erste einsetzen, was passt. Hierfür ist Backtracking die einfachste Lösung: Wir nehmen das erste Wort, welches in eine Lücke passt, speichern es ab und geben die Wörterliste ohne das gerade eingesetzte Wort weiter, und versuchen es bei der nächsten Lücke. Dies machen wir, bis (1) die Liste Leer ist, und wir am Ende angekommen sind, dann haben wir das Rätsel gelöst, oder (2) wir kommen an einem Punkt an, wo wir in unserer Wörterliste kein passendes Wort für die entsprechende Lücke haben. Wenn wir an (2) ankommen, dann gehen wir einen Schritt zurück, und überprüfen, ob wir in die vorherige Lücke auch ein anderes Wort einsetzen können. Dabei unterscheiden wir zwischen (A) ein anderes Wort passt in die Lücke, dann gehen wir wieder eine Lücke weiter, und (B) wir finden kein anderes passendes Wort für die vorherige Lücke, dann gehen wir wieder eine Lücke weiter zurück, und überprüfen, ob wir dort ein anderes Wort einsetzen können, solange, bis wir eine Lücke finden, wo wir auch ein anderes Wort aus der Liste einsetzen können. Dies machen wir, bis wir eine Lösung finden, oder wir zurück zur ersten Lücke kommen, und schon alle Wörter hier ausprobiert haben, dann gibt es keine Lösung.

Skizze der Lösungsidee



Umsetzung

Für die Umsetzung habe ich Java verwendet, da Java OOP unterstützt. Unterteilt ist mein Programm in 3 Klassen: Main, StartSolver, und SolveRiddle. Die Main Klasse dient nur als Entry-Point, wobei ein StartSolver Objekt erstellt wird. Der Konstruktor von StartSolver liest von STDIN den Dateinamen ein, und fügt die in der Datei enthaltenen Lücken und Wörter in ein zwei Arrays ein. Hier drauf wird ein Objekt SolveRiddle erstellt, welches im Konstruktor die Wörter und Lücken Arrays, und ein leeres Array für die Lösung mitgegeben bekommt. Darauf wird vom SolveRiddle Objekt die solve() Methode ausgeführt, wo der Backtracking Algorithmus implementiert ist. In der solve() Methode gehen wir mit einer For-Loop durch die Wörterliste durch und überprüfen für jedes Wort, ob es in die erste Lücke im (übergebenen) Array Lücke passt. Hierzu benutzen wir die Methode compare(String blank, String word): wenn compare() TRUE zurückgibt, dann passt das Wort in die Lücke. Darauf klonen wir die Lücken & Wort-Arrays mit der Methode cloneArray() und entfernen die jetzige Lücke, und das passende Wort, und erstellen ein neues Objekt SolveRiddle mit den geklonten Arrays, und rufen wieder solve() auf. Die Methode solve() gibt erst TRUE zurück, wenn wir keine Lücken mehr in unserem Array haben (wenn wir eine Lösung gefunden haben). Sobald der letzte Aufruf von solve() TRUE zurückgibt, fügen rekursiv alle vorher aufgerufenen SolveRiddle Objekte, die alle auf das TRUE warten, dass passende Wort ans Ende des am Anfang mitgegebenen Lösungsarray. Dieses enthält jetzt die Lösung in umgekehrter Reihenfolge, daher müssen wir dieses jetzt invertieren, wo drauf wir die Lösung für das Rätsel erhalten.

Kompilieren

Um das Programm zu Kompilieren, muss der Java Compiler installiert sein (**javac)**. Dann muss die Datei build.bat (Windows) ausgeführt werden, welche die .class Dateien in den Ordner „./executable“ legt.

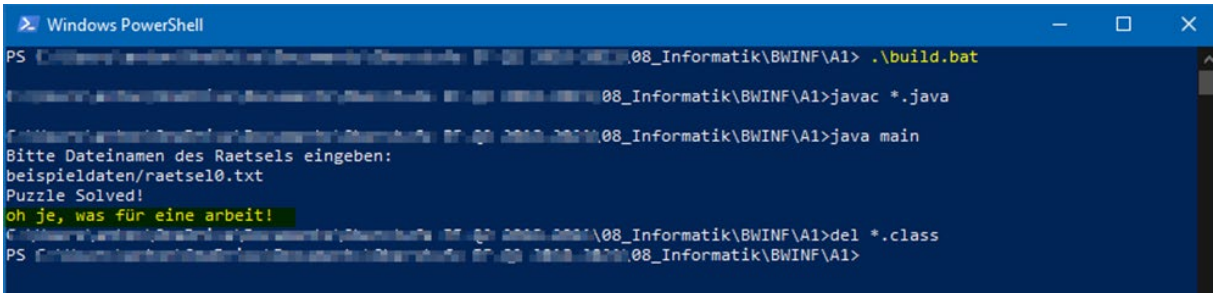
Beispiele

Ausführen von „cd executable; java Main“ in einem CMD und das Eingeben von entweder:

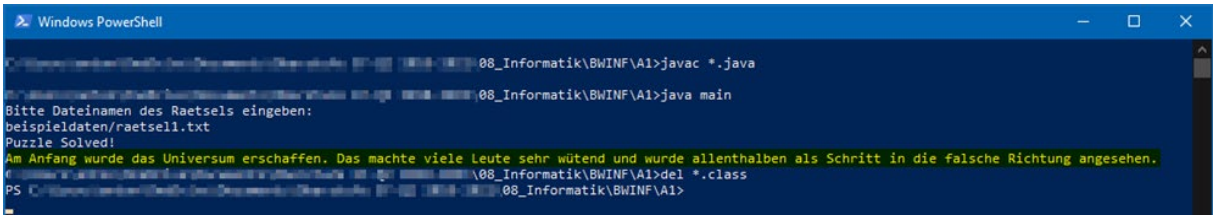
1. ../beispieldaten/raetsel0.txt
2. ../beispieldaten/raetsel1.txt
3. ../beispieldaten/raetsel2.txt
4. ../beispieldaten/raetsel3.txt

5. ../beispieldaten/raetsel4.txt

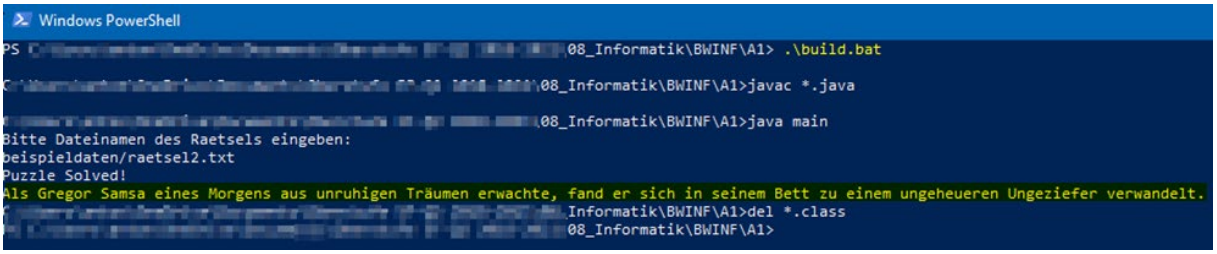
gibt zurück: (Bilder sind veraltet, aber Output ist der Gleiche)

1. 

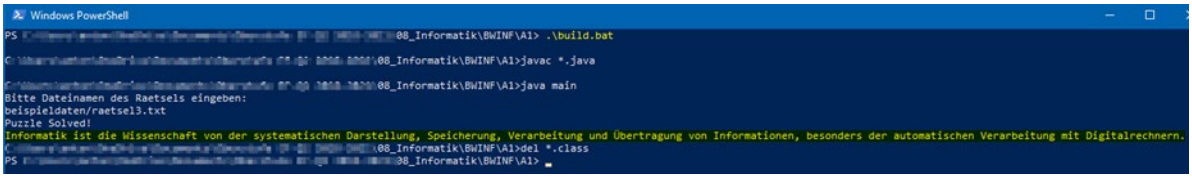
(oh je, was für eine arbeit!)

2. 

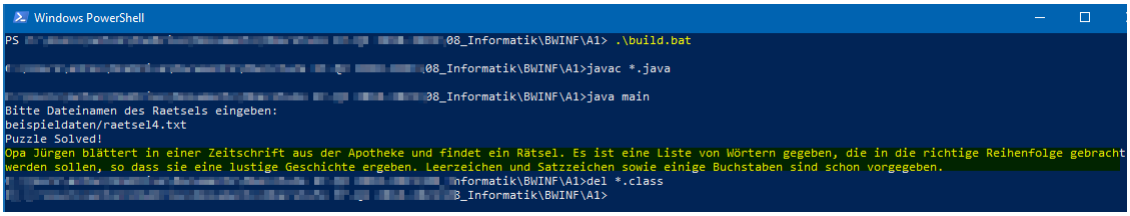
(Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.)

3. 

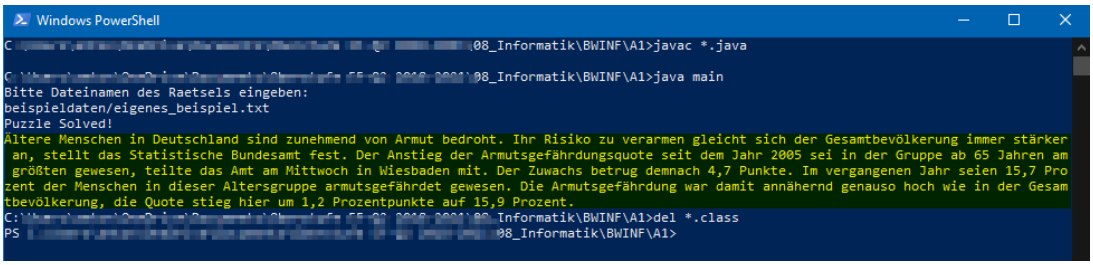
(Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.)

4. 

(Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern.)

5. 

(Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.)

6. 

Eigenes Beispiel – „beispieldaten/eigenes_beispiel.txt“: (Ältere Menschen in Deutschland sind zunehmend von Armut bedroht. Ihr Risiko zu verarmen gleicht sich der Gesamtbevölkerung immer stärker an, stellt das Statistische Bundesamt fest. Der Anstieg der Armutsgefährdungsquote seit dem Jahr 2005 sei in der Gruppe ab 65 Jahren am größten gewesen, teilte das Amt am Mittwoch in Wiesbaden mit. Der Zuwachs betrug demnach 4,7 Punkte. Im vergangenen Jahr seien 15,7 Prozent der Menschen in dieser Altersgruppe armutsgefährdet gewesen. Die Armutsgefährdung war damit annähernd genauso hoch wie in der Gesamtbevölkerung, die Quote stieg hier um 1,2 Prozentpunkte auf 15,9 Prozent.)

Grenzen und Optimierung

Solange es eine eindeutige Lösung gibt, und das Rätsel in dem Format wie den Beispielen gestellt ist, sollte dieses Programm die Lösung finden, ohne Ausnahmen. Trotzdem muss man bedenken, dass der Backtracking Algorithmus rekursiv ist, daher steigt mit der Länge des Rätsels exponentiell die Anzahl von Vergleichen, die Anzahl von Objekten, und somit Berechnungszeit und Speicherverbrauch. Hier könnten unterschiedliche Optimierungsschritte vorgenommen werden, z.B. könnten alle Lücken, wo nur ein Wort hereinpasst aussortiert werden, wodurch die Anzahl der Vergleiche stark abnimmt. Ein anderer möglicher Optimierungsschritt wäre, die Lücken nach der Länge zu gruppieren, und für jede Länge von Lücke ein eigenen „Backtracking-Baum“ zu machen. Somit unterteilen wir das Problem in mehrere kleineren Probleme, die wesentlich schneller gelöst werden können. Ich habe jetzt verzichtet, diese Optimierungen zu machen, da die Berechnungszeit für die angegebenen Beispiele auf meinem Rechner i.d.R. <1 Sekunde ist, und die Komplexität des Codes durch die Optimierungsschritte zunehmen würde.

Quellcode

solve(): Boolean

```
public boolean solve(){
    for(int i = 0; i < words.size(); i++) /* Hier gehen wir durch jedes Wort im Array einmal durch */
    {
        /* Vergleich von der jetzigen Lücke mit dem Wort */
        if(compare(blanks.get(0),words.get(i)))
        {

            /* wir klonen die Arrays und entfernen die Lücke und das passende Wort vom pool*/
            ArrayList<String> nBlanks = cloneArray(blanks);
            nBlanks.remove(0);
            ArrayList<String> nWords = cloneArray(words);
            nWords.remove(i);

            /* neues Objekt wird erstellt, wobei die geklonten Arrays uebergeben werden */
            SolveRiddle nriddle = new SolveRiddle(nBlanks,nWords,solution);

            /* aufruf von solve() */
            if(nriddle.solve())
            {

                /*
                 * Wenn wir diesen Punkt erreicht haben, hat der letze aufruf von solve() TRUE zurueckgegeben, und wir
                 * koennen unsere Loesung hinzufuegen
                 */

                /*
                 * Hier fügen wir noch, falls es ein Sonderzeichen am Ende der Lücke gab,
                 * dieses an mithilfe von der Methode checkForSpecial(), da die compare() funktion dieses Ignoriert hat
                 */
                solution.add(words.get(i) + checkForSpecial(blanks.get(0)));
                return true;
            }
        }
    }
    /*
     * Überprüfen, ob wir die Lösung haben (wenn keine Lücken mehr übrig sind)
     */
    if(blanks.size() == 0)
    {
            System.out.println("Puzzle Solved!");

            /*
             * Mit diesem TRUE wird die kaskade ausgelöst, wodurch alle vorherigen SolveRiddle Objekte
             * ihre lösung in das Lösungsarray hinzufügen
             */
            return true;
    }
    /*
     * Wenn wir in diesem SolveRiddle Objekt kein passendes Wort für die Lücke gefunden haben,
     * und wir noch nicht am Ende sind, dann müssen wir Backtracken, und andere Wörter für die Vorherige Lücke ausprobieren
     * daher FALSE
     */
    return false;
}
```

compare(): boolean

```
    public boolean compare(String blank, String word)
    {

        /*
         * Wir überprüfen, ob die Lücke und das Wort gleich lang sind (ohne berücksichtigung von Sonderzeichen)
         */
        int blank_length = blank.length();
        int word_length = word.length();
        char blank_last = blank.charAt(blank_length-1);
        if(blank_last == '!' || blank_last == ',' || blank_last == ':' || blank_last == '.')blank_length--;
        if(word_length != blank_length)return false;

        /*
         * Übereinstimmung von Lücke und Wort überprüfen
         */
        for(int i = 0; i < word_length; i++){
            if( (blank.charAt(i) == '_') || (blank.charAt(i) == word.charAt(i)) ){} //wildcard or match
            else
            {
                return false;
            }
        }
        return true;
    }
```

cloneArray(): ArrayList<String> & checkforSpecial(): String

```java
/* Methode, die ein DeepCopy von einem StringArray machen */
public ArrayList<String> cloneArray(ArrayList<String>src){
    ArrayList<String>ret = new ArrayList<String>();
    for(int i = 0; i < src.size(); i++){
        ret.add(src.get(i));
    }
    return ret;
}

/*
 * Wenn ein Sonderzeichen am Ende von x vorhanden ist,
 * gebe es zurück (Für das Anhängen von Sonderzeichen bei dem Hinzufügen von Lösungen in solve()
 */
private String checkForSpecial(String x)
{
    char r = x.charAt(x.length()-1);
    if(r == '!' || r == ',' || r == ':' || r == '.')return Character.toString(r);
    return "";
}
```

Programm Input (StartSolver Klasse)

```java
private ArrayList<String> luecken = new ArrayList<String>();
private ArrayList<String> woerter = new ArrayList<String>();
private ArrayList<String> loesung = new ArrayList<String>();
private Scanner input = new Scanner(System.in, "utf-8");

/* String aufteilen nach leerzeichen und Array a anhängen */
public void tokenizeAndAddToArray(String str, ArrayList<String> a)
{
    String[] tokens = str.split(" ");
    for(String token : tokens)a.add(token);
}

/* Input von Stdin einlesen */
public String readInput(String prepend) throws IOException
{
    System.out.println(prepend);
    return input.nextLine();
}

/* Scanner Objekt von Datei öffnen */
public Scanner getFileScannerHandle(String filename)
{
    Scanner reader = null;
    try
    {
        reader = new Scanner(new FileInputStream(filename), "utf-8");
    }
    catch(FileNotFoundException e)
    {
        System.out.println("Cannot open File, maybe doesnt exist?");
        System.exit(1);
    }
    return reader;
}

/* Konstruktor, von main() aufgerufen, der den Löseprozess startet */
public StartSolver() throws IOException
{
    Scanner reader = getFileScannerHandle(readInput("Bitte Dateinamen des Raetsels eingeben:"));

    tokenizeAndAddToArray(reader.nextLine(),luecken);
    tokenizeAndAddToArray(reader.nextLine(),woerter);

    SolveRiddle rid = new SolveRiddle(luecken,woerter,loesung);

    if(rid.solve())
    {
        /*
         * Wichtig! Wir müssen das Lösungsarray Invertieren,
         * da die SolveRiddle Objekte von LetzeLücke -> ErsteLücke die Lösungen anfügen
         */
        Collections.reverse(loesung);
        for(String x : loesung)System.out.print(x + " ");
    } else {
        System.out.println("Keine Loesung gefunden!");
    }
}
```