Aufgabe 4: Auto-Scrabble

Team: “throw new Exception();”
Einsendenummer: ???

23. November 2017

Inhaltsverzeichnis

Lösungsidee

Der Plan war es zunächst das Problem umfangreicher zu lösen, als es gefordert war.

Statt zu bestimmen, ob es eine Kombination von Kennzeichen gibt und dann genau eine auszugeben, sollten alle Kombinationen ermittelt werden. Der Algorithmus dafür funktioniert, auf Grund des benötigten Arbeitsspeichers ist er für längere Worte jedoch unbrauchbar. Folglich wurde er so modifiziert, dass nur die nötigsten Lösungen berechnet werden. Ich werde in diesem Abschnitt den umfangreicheren Algorithmus erwähnen und bei der Umsetzung genauer auf die Abweichung eingehen.

Außerdem gilt es zu sagen, dass es durchaus andere, ebenfalls rekursive Algorithmen gibt, die ohne Arbeitsspeicherprobleme alle Lösungen finden, diese leiden jedoch oftmals unter Laufzeitproblemen, da viele Dinge mehrfach berechnet werden müssen.

Gegeben sind zwei Listen mit Zeichenfolgen: eine enthält alle gültigen Kürzel und die andere die zu schreibenden Worte.

Wenn versucht wird Worte zu schreiben, muss unterschieden werden, ob nur eines oder mehrere Kennzeichen genutzt werden dürfen. Dabei passen auf ein einzelnes Kennzeichen nur maximal fünf Buchstaben und für zwei Kennzeichen muss das zu schreibende Wort mindestens vier Buchstaben lang sein, da pro Kennzeichen zwei Buchstaben die Mindestlänge darstellen und jedes Kennzeichen komplett ausgefüllt werden muss.

Um ein Wort auf ein Kennzeichen zu schreiben, werden aus der Kürzel-Liste alle Kürzel gesucht, deren Buchstaben mit den ersten Buchstaben des zu schreibenden Wortes übereinstimmen. Anschließend werden alle gefundenen Anfänge überprüft. Damit ein Anfang gültig ist, muss der Rest des Wortes genau einen oder genau zwei Buchstaben lang sein. Dabei ist zu beachten, dass Umlaute im Rest zu zwei Buchstaben, also AE, OE oder UE werden. Zurückgegeben werden alle Kombinationen von Anfang und Rest, die diese Bedingungen erfüllen.

Wenn es darum geht ein Wort auf mehrere Kennzeichen zu schreiben, ist eine rekursive Lösung angemessen. Sollte das zu schreibende Wort kürzer als drei Buchstaben lang sein, so wird versucht es auf ein einzelnes Kennzeichen zu schreiben. Das ist der elementare Fall.

Für alle Worte, die mindestens drei Buchstaben lang sind werden nach und nach alle Lösungen für die ersten n Buchstaben ermittelt und zwischengespeichert. (Auch hierbei gilt es zu erwähnen, dass Umlaute zwar für zusätzliche Buchstaben im Endergebnis sorgen, aber in Bezug auf n keinen Unterschied machen.) Dabei wird zunächst immer versucht diese n Buchstaben auf ein Kennzeichen zu schreiben. Das schlägt jedoch meistens fehl. Anschließend wird der aktuelle Abschnitt (n Buchstaben lang) in zwei Teile gespalten: die letzten k Buchstaben werden abgetrennt. Eine gültige Lösung setzt sich somit aus der Lösung für den Abschnitt von 0 bis n – k und den Abschnitt von n – k bis n zusammen.

Sollte es für den ersten Teil keine Lösung geben oder dieser zu kurz sein, braucht für den zweiten keine Lösung gesucht werden.

Für den zweiten Teil werden alle Längen von k=2 bis k=5 ausgetestet, da diese auf ein Kennzeichen passen. In jedem dieser Fälle wird nun versucht den zweite Abschnitt mit einem einzelnen Kennzeichen zu lösen. Jede erhaltene Lösung wird dann mit allen Lösungen des passenden ersten Abschnitt kombiniert. Auf diesem Wege werden alle Lösungen für den aktuellen Gesamtabschnitt gefunden und abgespeichert. Diese können später für größere Abschnitte weiterverwendet werden.

Nach einiger Zeit wird dann der Abschnitt der Länge n gelöst, insofern es eine Lösung gibt.

Umsetzung

Der oben beschriebene Algorithmus wurde in C# implementiert. Im Abgabe-Verzeichnis finden Sie vier Projekte. Eines enthält den Algorithmus zum Finden aller Lösungen, zwei weitere sind darauf spezialisiert nur eine Lösung pro Wort zu finden, wobei diese sich darin unterscheiden, dass das eine Programm einzelne Dateien und das andere ganze Verzeichnisse mit Textdateien einlesen kann. Das vierte Projekt kann in einer einzelnen Datei alle Worte (bzw. Zeilen) finden, die sich nicht nach dem gefoderten Schema darstellen lassen. Diese Variante wurde zum Lösen der zweiten Aufgabe genutzt. Mehr dazu steht im Abschnitt „Beispiele“.

Ich werde beim Beschreiben der Umsetzung nur zwischen den zwei verschiedenen Algorithmen zum Finden aller bzw. genau einer Lösung unterscheiden, da die Input- und Output-Behandlung keine Rolle für die eigentliche Lösung des Problems spielt.

Zu Beginn werden die Kürzel in eine Kürzel-Liste und die Worte in eine Wort-Liste eingelesen.

Dann wird mit Hilfe einer for-Schleife jedes eingelesene Wort bearbeitet. Bearbeiten heißt in diesem Fall, dass die Funktion zum Finden einer Lösung mit beliebig vielen Kennzeichen ausgeführt wird, das Ergebnis wird gespeichert.

Diese Funktion gibt für alle Worte, die kürzer sind als drei Zeichen, das Ergebnis zum Finden von Lösungen mit nur einem Kennzeichen zurück.

Diese Funktion wiederum findet alle Kürzel, deren Buchstaben mit den ersten Buchstaben des Wortes übereinstimmen und speichert diese. Die gefundenen Kandidaten werden anschließend dahingehend überprüft, dass der Rest des Wortes, nur ein bis zwei Buchstaben lang sein darf, wobei hier die Regelung für Umlaute gilt. Alle Kandidaten, auf die das zutrifft, werden dann in einem Array gemeinsam mit dem Rest des Wortes zurückgegeben. Falls nur eine Lösung gesucht ist, wird der erste Kandidat genommen, mit dem Rest gepaart und zurückgegeben. Dabei steht also die erste Stelle für das Kürzel und die Zweite für die nötigen Buchstaben um das Wort zu vervollständigen.

Zurück in der Funktion für Lösungen mit beliebig vielen Kennzeichen wird, falls das Wort länger als zwei Buchstaben lang ist, ein Array, dessen Länge der Länge des Wortes entspricht, erstellt. Jede einzelne Stelle enthält eine Liste für Lösungen für den Teil des Wortes von Stelle 0 bis zur Stelle des Arrays. Dann werden

in einer for-Schleife von 0 bis n – 1 (n = Länge des Wortes) alle möglichen Abschnitte des Wortes durchgegangen. In jedem Abschnitt wird versucht Lösungen mit nur einem Kennzeichen zu finden. Sollte das gelingen, geht die Funktion zum Finden von nur einer Lösung zum nächst größeren Abschnitt über, da nur relevant ist, ob es eine Lösung für den Abschnitt gibt oder nicht. Falls keine derartige Lösung gefunden wurde oder alle Lösungen erforderlich sind, werden Segment-Längen von 2 bis einschließlich 5 durchsucht.

Die Idee den Abschnitt in einen kleineren Abschnitt und ein Segment der o.g. Länge zu unterteilen wurde bereits beschrieben. Sollte eine Lösung für den kleineren Abschnitt existieren, so wird versucht eine Lösung mit nur einem Kennzeichen für das Segment zu finden. Im Folgenden unterscheiden sich die Algorithmen für eine/mehrere Lösung(en):

Sollte nur eine Lösung gefordert sein, so wird überprüft, ob das Ergebnis der Einzelkennzeichen-Funktion ein Element enthält. Wenn ja, wird dieses Ergebnis in einem neuen Array mit der bereits abgespeicherten Lösung für den kleineren Abschnitt gepaart. Auf diese Weise entsteht genau eine Lösung für den Abschnitt. In diesem Algorithmus ist außerdem zu beachten, dass nur so lange Segmente durchsucht werden, wie es für den aktuellen Abschnitt keine Lösung gibt.

In der vollständigen Lösungssuche hingegen wird jede gefundene Lösung für das Segment mit allen gespeicherten Lösungen für den kleineren Abschnitt gepaart. Somit werden alle Lösungen für den Abschntit gefunden.

Nachdem alle Abschnittlängen bearbeitet wurden, werden alle Lösungen für den Abschnitt mit der Länge des Wortes zurückgegeben.

Beispiele

Beispiele für diese Aufgabe zu bearbeiten stellte sich als relativ einfach heraus, da es keine spezielle Syntax zu beachten gab. Da das Programm alle Zeichen ignoriert, die keine Buchstaben sind, kann jede beliebige Textdatei eingelesen und bearbeitet werden. Zu beachten ist, dass, falls Umlaute o.Ä. verwendet werden, die Textdatei unbedingt in UTF-8 codiert sein muss.

Zunächst jedoch die Ergebnisse der Pflichtbeispiele (nur eine Lösung, ausführlicher im Anhang):

TIMO: keine Lösung gefunden

BIBER: [B-IB][E-R]

BUNDESWETTBEWERB: [B-UN][D-E][S-W][E-TT][B-E][W-E][R-B]

CLINTON: [C-LI][NT-ON]

DONAUDAMPFSCHIFFFAHRTSKAPITÄNSMÜTZE: keine Lösung gefunden

ETHERNET: [E-T][H-E][R-N][E-T]

INFORMATIK: [IN-FO][R-M][AT-IK]

LLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLLANTYSILIOGOGOGOCH:

[L-LA][N-F][A-I][R-P][W-L][L-G][W-YN][G-Y][L-L][G-O][G-E][R-Y][C-H][W-YR][N-D][R-OB][W-L][L-L][L-A][N-TY][S-I][L-IO][G-O][G-O][G-O][C-H]

RINDFLEISCHETIKETTIERUNGSÜBERWACHUNGSAUFGABENÜBERTRAGUNGSGESETZ:

keine Lösung gefunden

SOFTWARE: [SO-FT][W-A][R-E]

TRUMP: [TR-U][M-P]

TSCHÜSS: [TS-C][H-UE][S-S]

VERKEHRSWEGEPLANUNGSBESCHLEUNIGUNGSGESETZ:

[VER-KE][H-R][S-W][E-G][E-P][L-A][N-UN][G-S][B-E][S-C][H-L][E-U][N-I][G-U][N-G][S-G][E-S][E-TZ]

Anschließend habe ich mir mehr oder weniger zufällig Bücher von http://gutenberg.org besorgt. Diese wurden in Textdateien abgespeichert (s. Anhang). Fünf davon sind in deutscher Sprache, zwei sind auf Englisch.

Die output-Dateien können im Anhang begutachtet werden. Es ist jedoch festzustellen, dass erstaunlich viele Zeilen eine Lösung haben.

Zuletzt fiel mir auf, dass in der Aufgabenstellung gefordert wurde zwei-, drei- und vierstellige Worte zu finden, die sich nicht mit Kennzeichen darstellen lassen. Dafür habe ich ein Projekt so modifiziert, dass die Lösungen nicht ausgegeben werden, sondern nur dann ein Eintrag vorgenommen wird, wenn keine Lösung gefunden werden konnte.

Da ich keine Textdatei mit allen (bzw. vielen) zwei- bis vierstelligen Worten gefunden habe, wurde eine wörterbuchartige Datei benötigt. Das beste Ergebnis nach nicht allzu langem Suchen im Internet war folgende Statistik der Universität Leipzig, welche in absteigender Reihenfolge die Zehntausend meistgenutzten Worte der deutschen Sprache auflistet. (http://pcai056.informatik.uni-leipzig.de/downloads/etc/legacy/Papers/top10000de.txt) Wendet man nun das modifizierte Programm auf die Datei an, erhält man 814 Worte, die sich nicht darstellen lassen.

Um einerseits Abkürzungen, wie „s.“ zu entfernen und anderseits nur kurze Worte aufzulisten muss eine weitere Modifikation vorgenommen werden. Nun wird auch die Länge des Wortes überprüft. Also werden nur die Worte gespeichert, die 2 bis 4 Buchstaben lang sind.

Somit wurde 107 Worte gefunden. Diese sind in der Datei „Aufgabe 4.2 Lösung.txt“ im Anhang aufgelistet.

Beispielsweise: Yen, Club, Tanz, Tier, Show, Uwe

Quellcode

Zunächst die wichtigsten Abschnitte: Die vier verschiedenen Lösungsfindungsalgorithmen, je zwei für einzelne / mehrere Kennzeichen, je zwei für eine Lösung / alle Lösungen.

Diese Algorithmen sind, unabhängig davon, ob sie verwendet werden, in allen abgegebenen Projekten zu finden.

```
        static void Main(string[] args)
        {
            string tempPath = InputPath();
            Console.WriteLine("");
            int lastUpdate = 0;

            for (int i = 0; i < words.Count(); i++)
            {
                results.Add(FirstSolution(words[i]));
                lastUpdate = UpdateProgress(i, lastUpdate);
            }

            string returnPath = ReturnPath(tempPath);
            ReturnResults(returnPath);

            Console.WriteLine("The output was saved to: " + returnPath);
            Console.ReadKey();
        }
```

```
        static List<string[]> FirstSolution(string word)
        {
            List<string[]> solutions = new List<string[]>();
            char[] wordChar = word.ToCharArray();
            int length = wordChar.Count();
            if (length < 3) { return FirstShortSolution(word); }
            else
            {
                List<string[]>[] solutionsAt = new List<string[]>[length];
                string wordAtCurrentLength = "";
                for (int currentLength = 0; currentLength < length;
                                                    currentLength++)
                {
                    solutionsAt[currentLength] = new List<string[]>();
                    wordAtCurrentLength += wordChar[currentLength];
                    solutionsAt[currentLength].AddRange
                            (FirstShortSolution(wordAtCurrentLength));

                    solutionsAt = UpdateFirstSolutionAtLength
                                  (solutionsAt, currentLength, word);
                }
                return solutionsAt[length - 1];
            }
        }
```

```
        static List<string[]>[] UpdateFirstSolutionAtLength
        (List<string[]>[] solutionsAt, int length, string word)
        {
            if (length > 2)
            {
                for (int currentSegmentLength = 2; currentSegmentLength < 6;
                                                    currentSegmentLength++)
                {
                    if (solutionsAt[length].Count() > 0) { break; }
                    else if (UnfittingSegmentLength(solutionsAt,
                                    currentSegmentLength, length)) { }
                    else
                    {
                        string currentSegmentWord = "";
                        for (int i = length - currentSegmentLength + 1;
                                                        i <= length; i++)
                        { currentSegmentWord += word[i]; }
                        List<string[]> solutionsForCurrentSegment
                        = FirstShortSolution(currentSegmentWord);
                        if (solutionsAt[length - currentSegmentLength].Count() > 0)
                        {
                            if (solutionsForCurrentSegment.Count() > 0)
                            {
                                if (solutionsAt[length].Count() == 0)
                                {
                                    solutionsAt[length].Add(mergeStringArrays
                                    (solutionsAt[length - currentSegmentLength][0],
                                    solutionsForCurrentSegment[0]));
                                }
                            }
                        }
                    }
                }
            }
            return solutionsAt;
```

Diese Methode wird in der Umsetzung nicht direkt erwähnt. Das hängt damit zusammen, dass sie direkt in die zuvor dargestellte integriert werden könnte. Ich bin jedoch der Meinung, dass diese Darstellung sowohl für mich, als auch für Sie, also Außenstehende, besser verständlich ist. In der Methode geht es lediglich darum für ein bestimmtes Wort bis zu einer bestimmten Länge eine Lösung zu finden. Der Algorithmus dazu wurde bereits erklärt.

```
        static List<string[]> FirstShortSolution(string word)
        {
            List<string[]> solutions = new List<string[]>();
            if (word.Length > 1 && word.Length < 6)
            {
                List<string> possibleStarts = new List<string>();
                foreach (string data in database)
                {
                    char[] dataChar = data.ToCharArray();
                    bool identicalStart = true;
                    for (int i = 0; i < word.Length && i < dataChar.Count(); i++)
                    {
                        if (!(word[i] == dataChar[i]))
                        {
                            identicalStart = false;
                            break;
                        }
                    }
                    if (identicalStart)
                    {
                        possibleStarts.Add(data);
                    }
                }
                foreach (string possibleStart in possibleStarts)
                {
                    string rest = Rest(word, possibleStart);
                    if (rest.Length > 0 && rest.Length < 3)
                    {
                        string[] solution = new string[2] { possibleStart, rest };
                        solutions.Add(solution);
                        return solutions;
                    }
                }
            }
            return solutions;
        }
```

Die letzten drei Methoden existieren ebenfalls in einer anderen Ausführung, dabei wird das Wort „First“ durch „Every“ ersetzt. Im Quelltext werden dann nur einige Bedingungen zum Abrechen der Berechnung entfernt, damit alle möglichen Lösungen gefunden werden.
Sollte widererwartend Interesse an den leicht veränderten Versionen bestehen, befinden sich diese in jedem Projekt direkt neben den hier dargestellten.

```
static string[] mergeStringArrays(string[] array1, string[] array2)
{
    string[] mergedStringArray = new string[array1.Count() + array2.Count()];
    for (int i = 0; i < array1.Count(); i++)
    {
        mergedStringArray[i] = array1[i];
    }
    for (int i = array1.Count(); i < mergedStringArray.Count(); i++)
    {
        mergedStringArray[i] = array2[i - array1.Count()];
    }
    return mergedStringArray;
}
```

Diese Methode wird in den oben gezeigten Funktionen aufgerufen und dient lediglich dazu zwei Lösungen zweier Bereiche in einer neuen Lösung aneinanderzureihen.

Alle anderen Methoden unterscheiden sich zwischen den Projekten und/oder sind für die Dokumentation nicht relevant. Sollte dennoch das Interesse bestehen, finden Sie die Projekte in den beiliegenden Ordnern. Außerdem finden sie dort kompilierte Dateien. Sollten sie vorhaben Beispiele zu testen, empfehle ich, dass Sie den string in der Main-Methode mit dem Pfad zu ihrer kuerzelliste.txt ersetzen.