Aufgabe 1
Wörter aufräumen

Tassilo Tanneberger November 21, 2020

Theorie & Lösungsidee

Die Grundidee ist ziemlich simpel: wir suchen alle Wörter, die zu einem Lückenwort passen, heraus und sammeln diese in einer Liste. Das führen wir für jedes Lückenwort durch. Danach beginnen wir mit dem Lückenwort, welches die wenigsten passenden Wörter hat. Wir führen folgenden Schritt für jedes passende Wort aus. Wir löschen das Wort einmal aus den möglichen Wörtern anderer Lückenwörter, weil das Wort nun genutzt wird. Danach beginnt der Prozess wieder bei der Suche des Lückenwortes mit minimaler Anzahl von Auswahlmöglichkeiten.

Anmerkung zur mathematischen Formulierung: l ist ein Lückenwort aus der Menge aller Lückenwörter L und w ist ein mögliches angegebenes Wort aus W. Die Funktion $\Psi(l, w) \in \{0, 1\}$ trifft eine Aussage darüber ob ein Wort w zu einen Lückenwort l zugeordnet werden kann.

$$f(l) : L \to \{W\}$$
$$f(l) = \{w_i \in W |\ \Psi(l, w_i)\ i = (1, \ldots, |W|)\}$$
$$f(l)\ \ldots\ \textit{Alle Wörter, die zu einen gegebenen Lückenwort } l \textit{ passen}$$

Die Funktion $\Psi(l, w)$ schaut ob zu einem gegebenem Wort einen gegebenes Lückenwort passt. Ein nicht gesetzter Buchstabe eines Lückenwortes wird mit einem "_" gekennzeichnet. Die Funktion hinter Ψ iteriert einfach über die Buchstaben in dem Wort und dem Lückenwort und vergleicht die Buchstaben. Wenn er ein "_" findet wird der Buchstabe aus w ignoriert. Nur wenn jeder Buchstabe aus w mit jedem Symbol aus l übereinstimmt, wird 1 zurückgegeben.

$f(l)$ spielt eine wichtige Rolle, da es alle Wörter aussortiert, die nicht zu einen gegebenen Lückenwort l passen. Somit kann man Lückenwörter priorisieren abhängig davon, wie viele Wörter noch vorhanden sind, die zu dem Lückenwort passen ($|f(l)|$). Mathematisch betrachtet ist $f(l)$ eine Mengenfunktion.

Nun müssen den Algorithmus nur noch formal definieren. Ich habe den Algorithmus in zwei Funktionen gespalten. Die erste Funktion (Find-Min-Size(W, L)) sucht das Lückenwort heraus, was die geringste Auswahl an passenden Wörtern hat und somit als nächstes ein Wort zugewiesen bekommt. Die zweite Funktion (Recursive-Search(W, L, C)) generiert über einen Backtracking-Ansatz alle möglichen Sätze.

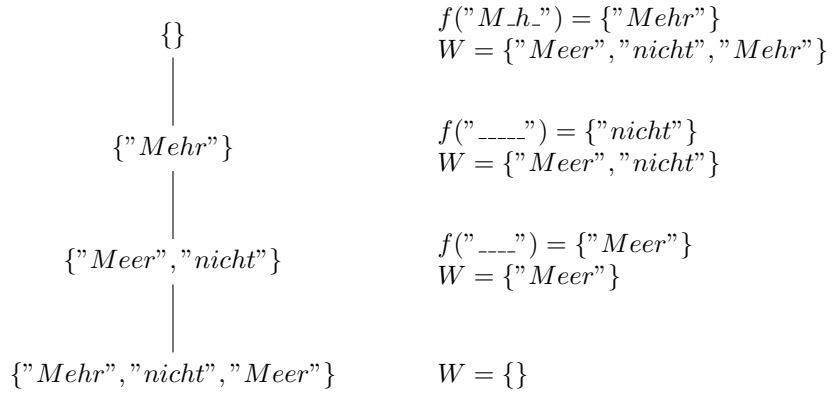
Algorithm 1 Finden möglicher Zuordnungen

```
1:  K ... Lösungsmenge
2:  function FIND-MIN-SIZE(W, L)
3:      λ ... MIN VALUE
4:      I ... INDEX
5:      for i = (1, ..., |L|) do
6:          if |f(l_i)| < λ ∧ |f(l_i)| ≠ 0 then
7:              λ ← |f(l_i)|
8:              I ← i
9:          end if
10:     end for
11:     return I
12: end function
13:
14: procedure RECURSIVE-SEARCH(W, L, C)
15:     I ← Find-Min-Size(W, L)
16:     if |K_I| = 0 then                          ▷ Abbruchbedingung
17:         if |C| ↔ |L| ∧ C ∉ K then
18:             K ← K ∪ {C}                        ▷ Mögliche Lösung gefunden
19:         end if
20:         return
21:     end if
22:     for w IN R_I do
23:         P ← W \ {w}
24:         C ← C ∪ {w}
25:         Recursive-Search(P, L, C)              ▷ Rekursiver Neuaufruf
26:     end for
27: end procedure
```

Nehmen wir ein kleines Beispiel und führen den Algorithmus damit durch.

- Lückenwörter $L = \{"M\_h\_", "\_\_\_\_\_", "\_\_\_\_"\}$
- Wörter $W = \{"Meer", "nicht", "Mehr"\}$

Von den gegebenen Lückenwörtern ist nur $l_3 =$ "____" uneindeutig: $|f("\_\_\_\_")| = 2$ (Verzweigung im Lösungsbaum). Somit wird l_3 als letztes ein Wort zugewiesen. Eine Verzweigung entsteht dann, wenn der Algorithmus sich nicht sicher ist, welches Wort er nun einsetzten soll: $|f(l)| > 1$.



Weitere Ideen Für eine besonders ambitionierte Lösung kann man auch die Funktion Ψ mit der Levenshtein-Distanz $d(l, w)$ ersetzen. Das würde den Vorteil mit sich bringen, dass man nicht mehr schaut ob ein gegebenes Wort auf ein Lückenwort passt, sondern wie ähnlich es dem Lückenwort ist.

Implementierung

Das Programm wurde in C++ umgesetzt. In diesem Abschnitt werden wir ausgewählte Quellcodestücke näher betrachten.

Code-Beispiel 1.)

```
unsigned int index = 0;
unsigned int min_value;

min_value = std::numeric_limits<unsigned int>::max();
for (unsigned int i = 0; i < possible_words.size(); i++) {
   if (min_value > possible_words.at(i).size() and not possible_words.at(i).empty()) {
      min_value = possible_words.at(i).size();
      index = i;
   }
}
```

Code-Beispiel 1 ist die Umsetzung der Funktion $Find - Min - Size(W, L)$, die aus dem Pseudo-Code bekannt ist. Wir benötigen zwei Variablen: `index` speichert, welches Lückenwort als nächstes besetzt werden muss und die Variable `min_value` sagt aus, wie viele Wörter in das Lückenwort passen, welches durch `index` referenziert wird.

Code-Beispiel 2.)

```
for (const auto &word : possible_words.at(index)) {
        // implicit deep copy
        std::vector<std::vector<std::string>> possible_words_copy = possible_words;

        // removing word from possible_words_copy
        for (std::vector<std::string> &vec : possible_words_copy) {
            auto iter_index = std::find(vec.begin(), vec.end(), word);
            if (iter_index != vec.end()) {
                vec.erase(iter_index);
                break;
            }
        }

        std::vector<std::string> result_copy = result; // making a deep copy implicitly
        result_copy.reserve(words_.size()); // already allocating memory
        result_copy[index] = word; // pushing the newly assigned word into the result container

        recursive(possible_words_copy, result_copy);
}
```

Das ist das Kernstück des Algorithmus. Hier entsteht einmal die Baumstruktur durch die For-Schleife, und auch der rekursive Neu-Aufruf ist in diesem Code-Beispiel untergebracht. Dieses Codesegment beginnt damit über alle möglichen Wörter für ein gewisses Lückenwort zu iterieren. In der Implementation wird W als `possible_words` bezeichnet und ist eine zweidimensionale Liste. Dabei repräsentiert `index` ein bestimmtes Lückenwort und die dazu gehörige Liste entspricht der oben definierten Funktion $f(l)$. Die Zeilen 5 bis 10 machen nichts anderes als das aktuelle Wort aus `possible_words_copy` zu entfernen. Danach fügen wir das aktuelle Wort dem Result-Container C hinzu. Jetzt haben wir W und C jeweils verändert und somit können wir die nächste Ebene des Baumes evaluieren.

Code-Beispiel 3.)

```
1  auto MatcherClass::match() -> result {
2      /*!
3       * The idea here is that we generate list of words that would fit that position and than start with the
       smallest
4       * list filling the string and removing the word from all list that contain the word
5       */
6      std::vector<std::string> result(words_.size());
7      std::vector<std::vector<std::string>> possible_words;
8
9      unsigned int iterator = 0;
10     for (const auto &pattern: text_) {
11         possible_words.emplace_back();
12         for (const auto &word: words_) {
13             if (MatcherClass::pattern_match(pattern, word)) {
14                 possible_words.at(iterator).push_back(word);
15             }
16         }
17         if (possible_words.at(iterator).empty()) {
18             return failure; // a pattern word has 0 fitting words therefore this is not solveable
19         }
20
21         iterator++;
22     }
23     recursive(possible_words, result);
24
25     return (corrected_text_.empty()) ? failure : success;
26 }
```

Durch den Konstruktor der `MatcherClass` wurden die Variablen `text_` und `words_` bereits gesetzt. In den Zeilen 9 bis 22 wird `possible_words_` (formal $f(l)$) erzeugt. Dabei wird über die Lückenwörter aus dem Lückentext iteriert und alle dazu passenden Wörter in einer Liste gesammelt.

Vorverarbeitung und Nachverarbeitung

Die Nutzereingabe besteht aus zwei Zeichenketten, aus denen zuerst alle Sonderzeichen wie "?", "!", ".", "," entfernt werden. Danach werden diese Strings in `std::vector<std::string>` gespalten, wobei Leerzeichen als Trennzeichen fungieren. Diese beiden Listen werden dann dem Algorithmus (`MatcherClass`) übergeben. Dieser produziert eine bestimmte Menge an Listen aus Zeichenketten. Die Listen müssen nun in der Nachverarbeitung wieder zu Zeichenketten zusammengesetzt werden. Zudem werden hier auch wieder die Sonderzeichen eingefügt.

Beispiel-Druchläufe

```
Aufgabe 1 des Bundeswettbewerbs Informatik 2020
================================================

Bitte geb nun den Lückentext ein. Buchstaben die weggelassen wurden werden durch einen _ Markiert.
_h __, _a_ __r ___e __b___!
Dein Eingegebener Lückentext: "_h __, _a_ __r ___e __b___!"
Text nach der Vorverarbeitung: "_h __ _a_ __r ___e __b___ "

Bitte gebe nun alle Wörter die in diesem Text vorkommen ein Wörter durch Whitespaces voneinander trennen
arbeit eine für je oh was
Dein Lückentext erwartet: 6 Wörter angegebene Wörter: 6
Deine Angegebenen Wörter:
[arbeit,eine,für,je,oh,was,]

Starting Matching ...
0 : oh je, was für eine arbeit!

Process finished with exit code 0
```

Figure 1: Konsolen-Output des Programms mit dem Beispiel aus der Aufgabenstellung (Rätsel 1)

```
Aufgabe 1 des Bundeswettbewerbs Informatik 2020
================================================

Bitte geb nun den Lückentext ein. Buchstaben die weggelassen wurden werden durch einen _ Markiert.
__s __e___ _a___ e____ _____n_ _u_ _________ ____m__ ________, _a__ _r s___ _n _____m ___t _u ____m __________ ______e___ _______e__.
Dein Eingegebener Lückentext: "__s __e___ _a___ e____ _____n_ _u_ _________ ____m__ ________, _a__ _r s___ _n _____m ___t _u ____m __________ ______e___ _______e__."
Text nach der Vorverarbeitung: "__s __e___ _a___ e____ _____n_ _u_ _________ ____m__ ________ _a__ _r s___ _n _____m ___t _u ____m __________ ______e___ _______e__ "

Bitte gebe nun alle Wörter die in diesem Text vorkommen ein Wörter durch Whitespaces voneinander trennen
er in zu Als aus Bett fand sich einem eines Samsa Gregor seinem Morgens Träumen erwachte unruhigen Ungeziefer verwandelt ungeheueren
Dein Lückentext erwartet: 20 Wörter angegebene Wörter: 20
Deine Angegebenen Wörter:
[er,in,zu,Als,aus,Bett,fand,sich,einem,eines,Samsa,Gregor,seinem,Morgens,Träumen,erwachte,unruhigen,Ungeziefer,verwandelt,ungeheueren,]

Starting Matching ...
0 : Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.

Process finished with exit code 0
```

Figure 2: Konsolen-Output des Programms mit dem Beispiel aus Rätsel 2 von der BWINF-Website