Übung 05: Dynamisches Programmieren

Programmiertechniken in der Computerlinguistik II, FS 16

Abgabetermin: 19.05.2016, 18:00 Uhr

Hinweise zur Abgabe

- Gib jedes Python-Programm in einer eigenen Datei ab, welche den Namen *aufgabe0x_vorname_nachname.py* trägt und **ausführbaren** Python-Code enthält.
- Geize nicht mit Kommentaren direkt im Programm-Code, wo Erläuterungen angebracht sind. Umfangreiche Erklärungen werden hingegen besser in einer separaten README-Datei mitgeliefert (vorzugsweise Plain-Text oder PDF).
- Alle Antworten, welche keine Python-Skripte sind, sollen als **eine einzige Datei** (Plain-Text oder PDF) mit dem Namen *vorname_nachname.pdf (.txt)* abgegeben werden.
- Um das Hochladen der Abgabe auf OLAT zu erleichtern, kannst du die Dateien mit `zip` oder `tar` (oder einem anderen verbreiteten Format) archvieren / komprimieren.

1 Längste gemeinsame Teilfolge

a) In der Vorlesung wurde ein Algorithmus vorgestellt, welches das Problem der längsten gemeinsamen Teilfolge (Longest Common Subsequence, LCS) löst. Wende diesen Algorithmus nun **manuell** an, um die längste gemeinsame Teilfolge der Wörter 'comics' und 'cosmic' zu finden. Bestätige deine Lösung, indem du die vollständige Matrix, welche dieser Algorithmus erzeugt, angibst und den Pfad darin markierst, welcher zur Rekonstruktion des Ergebnisses genutzt wird. Wähle einen der vorgestellten Ansätze (top down/bottom up) für deine Lösung.

b) Was ist die längste anzunehmende Laufzeit deines Lösungsansatzes (O-Notation)? In welchem Fall würde die längste anzunehmende Laufzeit (*worst case*) eintreffen?

c) Erkläre in wenigen Sätzen, in welcher Hinsicht dieser Lösungsansatz 'dynamisch' ist.

2 Levensthein-Distanz

Aus der Vorlesung kennst du die Berechnung der Levenshtein-Distanz auf buchstäblicher Ebene. In dieser Übung beschäftigen wir uns mit der Levenshtein-Distanz auf wörtlicher Ebene. Mit der Levenshtein-Distanz auf wörtlicher Ebene kann die Anzahl Operationen, um einen Satz `A` in einen Satz `B` umzuwandeln, berechnet werden. Folgend sind die Operationen und ihre Kosten definiert, welche du für diese Übung anwenden sollst:

1. Eine Interpunktion einfügen: 0.1
   Alles andere einfügen: 3.0
2. Eine Interpunktion entfernen: 0.1
   Alles andere entfernen: 3.0
3. Element `x` mit Element `y` ersetzen:

- `x` und `y` sind beides Interpunktionen: 0.1
- `x` und `y` sind beides Zahlen: 4.0
- `x` und `y` sind beides Wörter (*'abc3'* gilt als ein Wort): 1.3
- `x` und `y` sind nicht vom gleichen Typ: 16.0

Beispiel:

- Satz `A`: Vladimir Levenshtein übernahm dies im Jahre 1960.
- Satz `B`: Vladimir Iosifovich Levenshtein entwickelte dies im Jahre 1965.
- **3.0** ('Iosifovich' einfügen) + **1.3** ('übernahm' mit 'entwickelte' ersetzen) + **4.0** ('1960' mit '1965' ersetzen) = **8.3**

a) Folgend sind Satz `A` und Satz `B`. Berechne die Kosten der Operationen (= die Distanz), um Satz `A` in Satz `B` umzuwandeln und notiere jeden Schritt wie oben veranschaulicht. (`A` soll am Schluss so aussehen wie `B`):

- Satz `A`: Computerlinguistik 2 ist spannend.
- Satz `B`: Computerlinguistik macht Spass und ist spannend!
- Wieso ist die Distanz 22.1 **nicht** die optimale Lösung? Erkläre in höchstens drei Sätzen.

b) Implementiere nun die Funktion `lev_word(A,B)`, welche die Levenshtein-Distanz auf Wort-Ebene von Satz `A` nach Satz `B` anhand der oben definierten Kosten berechnet. Überprüfe dein Skript mit den Sätzen von a).

c) Jetzt sollst du die Funktion `gen_edit_dist(A,B)` implementieren, welche eine erweiterte Form der Levenshtein-Distanz ist. Zusätzlich zu den drei Operationen (einfügen, entfernen, ersetzen) soll nun untenstehende Operation mit ihren Kosten möglich sein. Anstelle der klassischen Levenshtein-Distanz kann nun die **generelle** Bearbeitungsdistanz ('general edit distance') berechnet werden.

- Zwei benachbarte Wörter transponieren: 0.4

Beispiel:

- Satz `A`: In Rätsel immer Yoda spricht.
- Satz `B`: Yoda spricht immer in Rätsel.

Generelle Bearbeitungsdistanz: **5.2**

d) *I wish you loved me.*
Verwende nun die generelle Bearbeitungsdistanz, um herauszufinden, welcher Satz im Brown Korpus (`nltk.corpus.brown.sents()`) diesem am ähnlichsten ist und was die generelle Bearbeitungsdistanz ist (die Berechnungen können eine Weile dauern).

Anmerkung: Zu dieser Übung ist ein Python-Skript Skelett unter dem Namen `aufgabe2.py` beigelegt, welches benutzt werden darf, um Aufgabe 2 zu lösen. Es ist aber nicht obligatorisch.

Reflexion/Feedback

a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen zusammen.

b) Wie viel Zeit hast du in diese Übungen investiert?