Parsing mit neuronalen Netzen: Einlesen der Daten

In den folgenden Übungen werden Sie einen Parser auf Basis von neuronalen Netzwerken implementieren. Der Parser orientiert sich an dem folgenden Artikel von Gaddy et al. (2018): `https://arxiv.org/pdf/1804.07853.pdf`

In der ersten Übung geht es darum, **Parsebäume einzulesen**, mit denen der Parser trainiert werden soll. Die Daten finden Sie in `http://www.cis.uni-muenchen.de/~schmid/lehre/data/PennTreebank.zip`.

Beispiel:

Für die Eingabe

```
(TOP(S(NP(NNP Ms.)(NNP Haag))(VP(VBZ plays)(NP(NNP Elianti)))(. .)))
```

soll zurückgegeben werden:

- die **Wortliste**: `["Ms.", "Haag", "plays", "Elianti", "."]`
- die **Konstituentenliste**: `[("TOP=S",0,5), ("NP",0,2), ("NNP",0,1), ("NNP",1,2), ("VP",2,4), ("VBZ",2,3), ("NP=NNP",3,4), (".",4,5)]`

  `TOP=S` und `NP=NNP` ergeben sich durch Eliminierung von Kettenregeln, die der Parser ja nicht verarbeiten kann.

Die folgenden Grammatikregeln erzeugen alle wohlgeformten Parsebäume:

tree → '**(**' **string** (**tree**+ | ' ' **string**) '**)**'
string → [**A-Za-z0-9...**]+ *(Folge beliebiger Zeichen außer '(', ')' und Whitespace)*

Schreiben Sie eine Funktion `read_tree` für die erste Regel und eine Funktion `read_string` für die zweite Regel im Stil eines Recursive-Descent-Parsers. `read_tree` analysiert den Parsebaum und ruft dabei `read_string` und rekursiv sich selbst auf. `read_tree` berechnet die Wortliste und die Konstituentenliste.

Nun müssen Sie noch Konstituenten mit gleichem Span zusammenfassen.

Als Nächstes schreiben Sie eine Funktion, welche aus der Wortliste und der Liste der Konstituenten wieder den **originalen Parsebaum** generiert und im Format des Originalbaumes ausgibt. Am besten nehmen Sie dafür auch wieder eine rekursive Funktion.

Zum Schluss schreiben Sie noch Code zum Testen des Modules, der mit folgender Zeile beginnt:

```
if __name__ == "__main__":
```

Diese Zeile verhindert, dass der Test-Code ausgeführt wird, wenn die Datei als Modul in ein anderes Programm importiert wird.

Aufruf: python tree-reader.py parsetrees.txt

Der Test-Code soll eine Folge von Parsebäumen aus einer als Argument übergebenen Datei einlesen, transformieren und rekonstruieren. Geben Sie für jeden Parsebaum aus:

- den Originalparse
- die Liste der Wörter
- die Liste der Konstituenten
- den wiederhergestellten Parsebaum

Vorüberlegungen

- Welche Argumente benötigen die Funktionen und was liefern sie zurück?
- Wie können Sie die Start- und Endposition einer Konstituente ermitteln?
- Wie können Sie mit zusätzlichen Leerzeichen in der Eingabe umgehen und Indexfehler beim Zugriff auf den Eingabestring vermeiden?
- Wie können Sie die Kettenregeln entfernen?
- Wie gehen Sie mit Fehlern in der Eingabe um?
- Wie kann der Parsebaum rekonstruiert werden?