Earley-Parser

In dieser Übung implementieren Sie einen Earley-Parser. Er liest eine Grammatik und ein Lexikon ein und analysiert dann tokenisierte Eingabesätze aus einer Datei (ein Satz pro Zeile). Der Parser gibt aus, ob der Satz grammatisch ist oder nicht.

Das **Startsymbol** der Grammatik ist das erste Symbol, das in der Grammatikdatei auftaucht.

Funktionen des Earley-Parsers

- **scan** liest das nächste Eingabewort w, schlägt seine Wortarten im Lexikon nach und ruft die Methode `add` für jede Wortart T auf, um eine Punktregel der Form $T \rightarrow w \cdot$ in die Chart einzutragen.
- **predict** wird mit einem Nichtterminal X und einer Position i als Argumenten aufgerufen und fügt für jede Grammatikregel der Form $X \rightarrow \alpha$ einen Eintrag $(X \rightarrow \cdot \alpha, i, i)$ zur Chart hinzu.
- **complete** wird mit einem Nichtterminal Y, einer Startposition m und einer Endposition e aufgerufen, sucht alle Charteinträge der Form $(X \rightarrow ... \cdot Y..., s, m)$ und vervollständigt sie.
- **add** wird von `scan`, `predict` und `complete` aufgerufen, um eine Punktregel in der Chart einzutragen. Wenn die Punktregel bereits in der Chart ist, wird nichts weiter getan. Wenn in der Punktregel auf den Punkt ein Nichtterminal folgt, ruft die `add`-Funktion die `predict`-Funktion mit diesem Nichtterminal auf. Wenn der Punkt ganz am Ende steht, ruft sie die `complete`-Funktion auf.

Programmaufruf: `python earley.py grammar.txt lexicon.txt sentences.txt`

Beispielgrammatik

```
S NP VP
NP NP PP
NP D N1
...
```

Beispiellexicon

```
saw V N
the D
...
```

Eine vollständige Beispielgrammatik finden Sie unter
`http://www.cis.uni-muenchen.de/~schmid/lehre/data/grammar.txt`
und ein Beispiellexikon unter
`http://www.cis.uni-muenchen.de/~schmid/lehre/data/lexicon.txt`

Vorüberlegungen

- Welche Aktionen werden beim Parsen des Satzes *I saw* nacheinander ausgeführt?
- Wie repräsentieren Sie am besten die **Chart**, damit alle Punktregeln mit derselben Endposition direkt nachgeschlagen werden können? Dies ist wichtig für eine effiziente Implementierung der `complete`-Funktion. Außerdem sollte der **minimale Speicherplatzbedarf** nicht quadratisch mit der Länge der Eingabe ansteigen. Auch sollte es in konstanter Zeit möglich sein, zu prüfen, ob ein Element in der Chart enthalten ist.
- Welche Repräsentation eignet sich für die Charteinträge? Wie repräsentieren Sie am besten die Information über die Position des Punktes, damit der Punkt mit minimalem Aufwand um eine Position weiterverschoben werden kann?
- Welche Datenstruktur eignet sich für das **Lexikon**? Die möglichen Tags zu einem Wort sollen ohne Suche direkt nachgeschlagen werden können.
- Wie kann die **Grammatik** repräsentiert werden? Für ein gegebenes Nichtterminal sollten alle Grammatikregeln mit diesem Nichtterminal auf der linken Seite direkt nachgeschlagen werden können.
- Wenn die `predict`-Funktion aufgerufen wird, nachdem sie zuvor schon einmal mit genau denselben Argumenten aufgerufen wurde, dann soll die Predict-Operation nicht erneut ausgeführt werden. Überlegen Sie sich einen Mechanismus, mit dem Sie das verhindern können.
- Wie erreichen Sie dasselbe für die `complete`-Funktion?

Schreiben Sie eine Klasse `Parser`, deren Konstruktor die Namen der Lexikondatei und der Grammatikdatei als Argumente erhält und Methoden zum Einlesen der Grammatik und des Lexikons aufruft. Außerdem implementieren Sie die vier Methoden `add`, `scan`, `predict` und `complete` und eine Methode `parse`, welche eine Liste von Wörtern als Argument erhält und `True` oder `False` zurückliefert, je nachdem, ob das Parsen erfolgreich war oder nicht. `parse` initialisiert die Chart, ruft die `predict`-Funktion mit dem Startsymbol auf und dann die `scan`-Funktion für jedes einzelne Wort. Zum Schluss prüft die Funktion, ob eine vollständige Analyse des Satzes gefunden wurde.

Implementieren Sie außerdem das Hauptprogramm des Parsers.

Weitere Aufgabe: Profiling

Wenn ein Programm zu langsam ist und deswegen seine Effizienz verbessert werden muss, ist ein Profiler ein nützliches Werkzeug. Ein Profiler ist ein Programm, welches ein anderes Programm bei dessen Ausführung überwacht und ermittelt, wieviel Zeit es bei der Ausführung welcher Funktion verbraucht. Meist sind es nur ganz wenige Funktionen, welche den Großteil der Rechenzeit verbrauchen, Diese können dann gezielt optimiert werden.

Sie sollen nun den Umgang mit dem Python-Profiler üben. Lassen Sie dazu Ihren Parser auf einem langen Satz laufen und erstellen Sie mit dem Modul `cProfile` Statistiken über die Laufzeiten der einzelnen Funktionen.

```
> python -m cProfile earley.py grammar lexicon sentences
```

Der Parser sollte einige Sekunden laufen. Wenn er kürzer läuft, hängen Sie einfach weitere PPs an den Satz an (I saw the man on the hill on the hill ...) und/oder weitere Sätze an die Datei.

In welcher Funktion verbraucht Ihr Programm die meiste Rechenzeit (ohne die Zeit, die in den dort aufgerufenen Unterfunktionen verbraucht wird)?

Schicken Sie Ihren Code, die Ausgabe des Profilings und die Antwort auf die obige Frage an `schmid@cis.lmu.de`.