Herausforderungen beim Parsen von PDF-Dateien

1. Aufbau von PDF-Dokumenten

PDF ist eine Seitenbeschreibungssprache basierend auf PostScript und beinhaltet Anweisungen wie Texte und Grafiken platziert werden sollen. Eine PDF-Datei ist also nicht mehr als eine Vektorgrafik, die weder semantische Auszeichnungen noch eine logische Strukturierung der Inhalte besitzt.

Aus was besteht also ein PDF-Dokument genau?

- *Encoded Characters:* Eine Folge von Bytes, welche die eigentlichen Zeichen repräsentieren
- *Font Data:* Eine Gruppe von Glyphen, die für die graphische Visualisierung einzelner Zeichen zuständig sind
- Einer Tabelle, die die kodierten Zeichen mit den Glyphen verbindet
- *Stream Objects:* Beinhalten Daten für die Darstellung von größere Inhalten

2. Probleme beim Parsen von PDF-Dateien

Zwar ist es möglich, Text aus PDF-Dateien zu extrahieren, allerdings wird man dabei vor einige Probleme gestellt:

- Gruppierung von Texten verläuft nicht nach inhaltlichen Aspekten
- Konvertierung von Schriften in das gewünschte Ausgabeformat
- Parsen von Tabellen, Kopf- und Fußzeilen sowie Grafiken und Bildern

3. Lösungsansätze für ein besseres Parsen

Eine Lösung, um das Parsen von PDF-Dokumente zu optimieren wäre, das Dokument zuerst in kleinere logische Elemente zu segmentieren. Je nach dem wie die einzelnen Elemente untereinander positioniert sind, werden sie zu logischen Gruppen verknüpft. Das heißt, der Inhalt wird nicht mehr nach dessen graphischen Anordnung im PDF-Dokument, sondern nach dessen semantischen Zusammenhängen geparst.

4. Tools zum Parsen von PDF-Dateien: PDFMiner

- Ausgelegt auf die Analyse und Auswertung von Texten in PDF-Dateien
- Versuch der Widerherstellung der logischen Strukturen in einem Dokument aufgrund der Positionierung der einzelnen Elemente
- Transformation in Formate wie HTML oder Plain Text

5. Beispielcode

An dem Beispielcode soll der Aufbau eines PDF-Dokuments nochmal verdeutlicht werden. Die aufgeführte Funktion bekommt alle Layout-Objekte einer einzelnen Seite übergeben und unterscheidet diese dann nach deren Inhalt:

```
# Die Funktion bekommt für eine einzelne Seite die gesamten Layout-Objekte
# übergeben. Daneben wird noch die Seitenzahl übergeben, um optional
# festzustellen auf welcher Seite welcher Inhalt steht.
def parse_lt_objs(lt_objs, page_number, text = []):
# Liste, in der der Text jedes Textobjektes (LTTextBox, LTTextLine) angehängt wird
    text_content = []

# In der for-Schleife warden die Layout-Objekte jeder Seite iteriert und
# unterschieden.
    for lt_obj in lt_objs:

# Ist das Objekt eine Instanz der Klasse LTTextBox oder LTTextLine wird der
# Text an die Liste angehängt.
        if isinstance(lt_obj, LTTextBox) or isinstance(lt_obj, LTTextLine):
            text_content.append(lt_obj.get_text())

# Ist das Objekt ein Bild kann man das optional in einem Ordner speichern
# und im Text kenntlich machen.
        elif isinstance(lt_obj, LTImage):
            pass

# Ist das Objekt eine Instanz der Klasse LTFigure wird die Funktion
# rekursiv aufgerufen; dabei wird das LTFigure-Objekt übergeben. Dieses
# kann nämlich selbst wieder andere Layout-Objekte enthalten.
        elif isinstance(lt_obj, LTFigure):
           parse_lt_objs(lt_obj, page_number, text_content)

# Am Ende der Funktion werden die Elemente der Liste zu einem String
# zusammengefügt und mit einer Leerzeile getrennt.
    return '\n'.join(text_content)
```