Institut für Angewandte und Numerische Mathematik
Sommersemester 2021
Dr. Daniel Weiß, M.Sc. Kevin Ganster

Einführung in Python

Abschlussprojekt

Eine einfache Suchmaschine

In diesem Projekt soll ein Programm entwickelt werden, das die Homepage der Mathematik-Fakultät nach Wörtern durchsuchen kann, die der Benutzer des Programms festlegt – so wie es Suchmaschinen wie Google oder Bing für das WWW anbieten.

Das Programm besteht aus drei Komponenten:

a) dem *Crawler*,

b) einer *Vorverarbeitung* und

c) der *Benutzerschnittstelle*.

Die Aufgabe des Crawlers besteht darin, die Seiten der Mathematik-Homepage systematisch herunterzuladen und die Beziehungen zwischen den Seiten zu ermitteln. Die Ergebnisse sollen in einer geeigneten Weise abgespeichert werden, sodass die Benutzerschnittstelle darauf zurückgreifen kann. Hier soll der Benutzer der Suchmaschine ein oder mehrere Suchwörter festlegen können, woraufhin das Programm die vom Crawler angelegte Datenbank durchsucht und eine Liste der Seiten ausgibt, die zu den Suchkriterien passen.

Von besonderer Bedeutung – insbesondere bei vielen Ergebnissen – ist die Reihenfolge, in der die Trefferseiten angezeigt werden. Dazu soll die Vorverarbeitung die vom Crawler gesammelten Daten auswerten und die Wichtigkeit mithilfe von einer Version des *Page-Rank-Algorithmus* ermitteln, die im folgenden erklärt wird.

Die Wichtigkeit einer Seite

Ein Netzwerk als Menge von sich gegenseitig verlinkenden Webseiten, kann als gerichteter Graph modelliert werden, bei dem jeder Knoten einer Internetseite und jede Kante einem Link entspricht, wie beispielsweise in Abbildung 1 gezeigt.

Um die Trefferseiten einer Anfrage sinnvoll zu sortieren, muss eine Suchmaschine die Wichtigkeit jeder Seite schätzen. Die populäre Suchmaschine Google verwendet dazu einen Algorithmus, der auf den folgenden Ideen beruht.

Seien $S_1, \dots, S_N$ Internetseiten, die sich gegenseitig verlinken. Jeder Link wird als Kante eines gerichteten Graphen von verlinkender zu verlinkter Seite aufgefasst. Zu

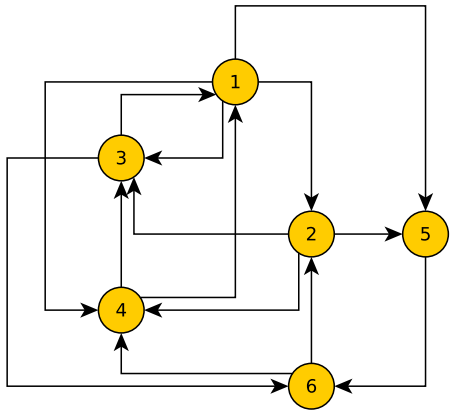

Abbildung 1: Ein Netzwerk mit 6 Seiten.

einer Seite S_n sei L_n die Anzahl ausgehender Links und

$$\mathcal{I}_n = \{j \in \mathbb{N} \colon S_j \text{ enthält Link auf } S_n\}$$

die Menge der Indizes von Seiten, die einen Link auf S_n enthalten.

Die Wichtigkeit x_n einer Seite S_n wird dann als gewichtete Summe festgelegt

$$x_n = \sum\nolimits_{j \in \mathcal{I}_n} \frac{x_j}{L_j},$$

d.h. jede Seite S_j, die auf S_n verlinkt, vergrößert die Wichtigkeit von S_n. Dabei ist der Beitrag umso größer, je weniger ausgehende Links S_j enthält und je wichtiger die Seite S_j selbst bewertet ist.

Diese Beziehungen zwischen den x_n führen auf ein Eigenwertproblem $A\mathbf{x} = \mathbf{x}$ mit $\mathbf{x}^\top = (x_1, \dots, x_N)$.

Für das Netzwerk in Abbildung 1 ergibt sich die Matrix

$$A = \begin{pmatrix} 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 1/4 & 0 & 0 & 0 & 0 & 1/2 \\ 1/4 & 1/3 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/3 & 0 & 0 & 0 & 1/2 \\ 1/4 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1 & 0 \end{pmatrix}.$$

Statt A betrachtet man die Matrix $M = (1-m)A + \frac{m}{N}B$ für $m \in (0, 1]$, wobei $B \in \mathbb{R}^{N \times N}$ die Matrix ist, die nur Einsen enthält, vgl. [1].

Es lässt sich zeigen, das M einen eindimensionalen Eigenraum zum Eigenwert 1 besitzt und dass sich dieser durch die Vektoriteration

$$\mathbf{x}^0 \in \mathbb{R}^N, \qquad \mathbf{x}^{k+1} := M\mathbf{x}^k, \quad k = 0, 1, 2, \ldots$$

mit Startwert $(\mathbf{x}^0)^\top = (1, \ldots, 1)$ approximieren lässt.

Arbeitsaufträge

a) Schreiben Sie ein Programm, das ausgehend von der Startseite http://www.math.kit.edu alle Seiten der Mathematik-Homepage ausliest, indem es alle *internen* Links auf der Seite ermittelt und die jeweils verlinkten Seiten herunterlädt. Dann soll auf jede der verlinkten Seiten das gleiche Verfahren angewendet werden, sodass rekursiv alle Seiten der Homepage durchlaufen werden.

Insbesondere sollen die Folgenden Teilprobleme gelöst werden:

(i) Es soll ein Modul ausgewählt werden, das den Quelltext einer Internetseite herunterladen kann.

(ii) Neben HTML-Seiten gibt es auch Links auf PDF-Dateien, Bilder, Word-Dokumente oder Videodateien. Diese Links sollen vom Crawler erkannt und ignoriert werden.

(iii) Mithilfe eines geeigneten Moduls sollen die Hyperlinks jeder Webseite im Quelltext identifiziert werden. Dabei ist zu beachten, dass es absolute und relative Hyperlinks gibt. Enthält beispielsweise eine Internetseite unter der Adresse http://www.abc.de/fgh/ijk den (relativen) Link `lmn`, so verweist dieser auf die absolute Adresse http://www.abc.de/fgh/ijk/lmn.

Hinweis: Stichworte: HTML-Parser oder reguläre Ausdrücke.

(iv) Jede Seite soll nur ein einziges Mal heruntergeladen und betrachtet werden. Startet man im Netzwerk aus Abbildung 1 bei Seite 1, führt einen diese auf die Seiten 2, 3, 4 und 5. Seite 3 verlinkt selbst wieder auf Seite 1, sodass das Programm erneut den Links dieser Seite folgen würde. Es soll eine Strategie implementiert werden, die solches Mehrfachauslesen verhindert, wobei trotzdem garantiert ist, dass jede Seite des Netzwerks besucht wird.

(v) Es soll ein Ablageformat entwickelt werden, mit dessen Hilfe die Ergebnisse eines Crawlerlaufs auf der Festplatte des Computers abgespeichert werden.

Hinweis: Legen Sie zum Testen Ihres Programms eine maximale Suchtiefe für den Crawler fest. Die Suchtiefe gibt dabei an, wie viele Links zwischen einer durchsuchten Seite und der Startseite maximal liegen dürfen.

b) Schreiben Sie ein Programm, welches die vom Crawler gesammelten Daten einliest und mithilfe des Page-Rank-Algorithmus die Wichtigkeit jeder der Seiten ermittelt und das Resultat in geeigneter Form abspeichert. Wählen Sie dabei $m = 0.15$ (dieser Wert wurde Berichten zufolge ursprünglich von Google verwendet).

Hinweis: Greifen Sie hierzu auf geeignete Teile von `numpy` *bzw.* `scipy` *zurück – Stichwort: dünn besetzte Matrizen.*

c) Implementieren Sie eine (textuelle) Benutzerschnittstelle, mit deren Hilfe der Benutzer die gesammelten Daten nach einem oder mehreren Wörter durchsuchen kann. Die Trefferseiten sollen in der von der Vorverarbeitung ermittelten Reihenfolge ausgegeben werden und außerdem soll ein kurzer Teil der Seite in der Umgebung der Treffer angezeigt werden.

Hinweis: Um die Volltextsuche zu beschleunigen, können Sie beispielsweise auf `concurrent.futures` *zurückgreifen.*

d) (Bonusaufgabe) Implementieren Sie eine grafische Benutzeroberfläche mithilfe von `PyQt`, die dieselbe Funktionalität wie die Nutzerschnittstelle unter (c) besitzt. Überlegen Sie sich eine übersichtliche Art der Ergebnispräsentation und zeigen Sie die Ergebnisse in einem `QListView`-Widget an.

Literatur

[1] Kurt Bryan and Tanya Leise. The $25,000,000,000 eigenvector: The linear algebra behind google. *Siam Review*, 48(3):569–581, 2006.

Material: ILIAS