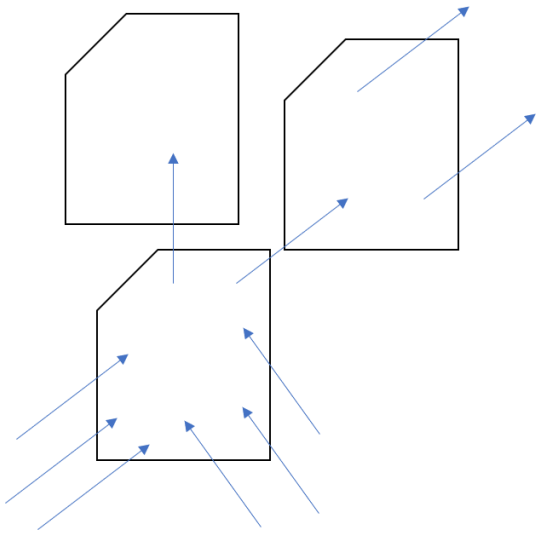
4 Semesterarbeit: Page-Rank

Information vorneweg: Die Vektoren tragen hier kein Vektorenzeichen über dem Buchstaben.

Verwendung von Page-Rank

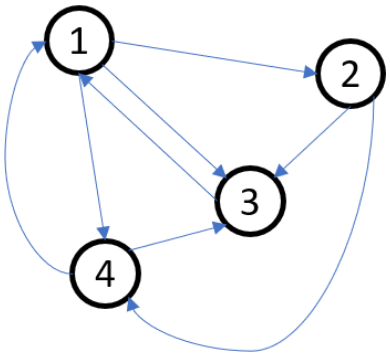
Der Page-Rank wird dazu verwendet Seiten zu gewichten und in einer Suchmaschine gemäss jener Gewichtung zu ordnen und aufzulisten. Ausschlaggebend für die Berechnung der Gewichtung ist die Verlinkungsstruktur von Seite zu Seite. Die Verlinkungen können wie Zitate angesehen werden Umso mehr Hyperlinks auf eine Seite zeigen, desto wichtiger scheint die Seite zu sein und desto wahrscheinlicher ist es, dass ein Surfer sich auf jener Seite befinden wird. Nebst dem eigentlichen Auffindern der relevanten Seiten aufgrund der Eingabe des Suchbegriffes kann so anhand des Page-Rank eine Rangliste erstellt werden. Anbei eine Illustration, die die Popularität der Seiten aufzeigt. Gemäss des Page-Rank Algorithmus ist es wahrscheinlicher, dass ein Surfer auf die Seite mit den sechs eingehenden Links gelangen wird als bei allen anderen präsentierten Seiten.



Abbild 1 Verlinkungsstruktur der Seiten

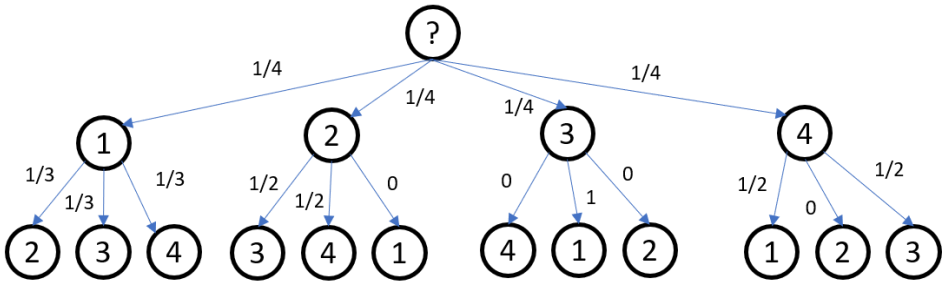
Berechnung von Page-Rank

Um nun den Page-Rank berechnen zu können, gilt es die Verlinkungsstruktur der Seiten anzusehen. Dazu gilt es die Idee an ein paar wenigen Seiten zu erklären, in diesem Falle sind es veir Seiten. Das Modell ist dann natürlich auf die Fülle des wirklichen Word Wide Web anwendbar.



Abbild 2 Ein gerichteter Graph

Die Knoten dieses gerichteten Graphen entsprechen den vier Seiten, die Pfeile entsprechen den Hyperlinks, sprich der Verlinkungsstruktur. Die Summe der Wahrscheinlichkeiten eines Random Surfer (ein Surfer der zufällig surft) besteht nun aus der Wahrscheinlichkeit, dass ein Surfer auf eine der vier Seiten als Ausgangslage gelangt und aus der Wahrscheinlichkeit von jener Seite her anhand der Verlinkungstruktur sich von Seite zu Seite weiterzubewegen. Die Ausgangslage entspricht also einem Viertel, da es ja vier Seiten sind. Von der ersten Seite aus geht es mit einer Wahrscheinlichkeit von einem Drittel zur zweiten Seite, zur dritten Seite und auch zur vierten Seite, das es sich ja um drei ausgehende Pfeile handelt. Dies kann so auch für die weiteren Knoten betrachtet und in einem Wahrscheinlichkeitsbaum zur Darstellung gebracht werden:



Abbild 3 Ein Wahrscheinlichkeitsbaum mit Kindeskinder

Hier soll die Idee der Wahrscheinlichkeitsberechnung anhand eines ersten Schrittes gezeigt werden, was den Kindern der Kinder entspricht:

Nach 1 zu gelangen: p(1) = ¼ * 0 + ¼ * 1 + ¼ * ½ = 3/8

Nach 2 zu gelangen: p(2) = ¼ * 1/3 + ¼ * 0 + ¼ * 0 = 1/12

Nach 3 zu gelangen: p(3) = ¼ * 1/3 + ¼ * ½ + ¼ * ½ = 1/3

Nach 4 zu gelangen: p(4) = ¼ * 1/3 + ¼ * ½ + ¼ * 0 = 5/24

Es wird schnell klar, dass diese Darstellung der Wahrscheinlichkeiten schnell unübersichtlich wird, da die Äste selbst wirder Äste tragen können. Daher bietet es sich an eine sogenannte Adjanzenzmatrix zu verwenden. Es handelt sich hierbei um eine quadratische Matrix, die den gerichteten Graphen repräsentiert. Die Anzahl Elemente entspricht der Multiplikation der Anzahl vorhandener Knoten, in diesem Falle sind das vier Knoten, also vier mal vier, ergibt sechzehn Elemente. Des Weiteren wird noch ein sogenannter Zustandsvektor p_0 gebraucht, der aus vier Zeilen und einer Spalte besteht, sprich die vier vorhandenen Knoten repräsentiert. Seine Elemente entsprechen den Wahrscheinlichkeiten der Ausgangslage, was bereits oben mit einem Viertel erwähnt wurde. Bei der ersten Multiplikation der Matrix mit dem Zustandsvektor erhalten wir die Wahrscheinlichekiten nach dem ersten Klick. Ziel ist es nun diese Multiplikation sooft zu wiederholen, bis ein stationärer Vektor erreicht wird, was einem Vektor entspricht, dessen Elemente unverändert bleiben (bei drei Dezimalstellen nach dem Komma gilt der Vektor als stationär). Bei diesem Beispiel soll dies bei einer Zahl von fünfzig erreicht werden:

$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 0 & 1 & 1/2 \\ 2 & 1/3 & 0 & 0 & 0 \\ 3 & 1/3 & 1/2 & 0 & 1/2 \\ 4 & 1/3 & 1/2 & 0 & 0 \end{array} \qquad p0 = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$$

Abbild 4 Adjazenzmatrix und Zustandsvektor

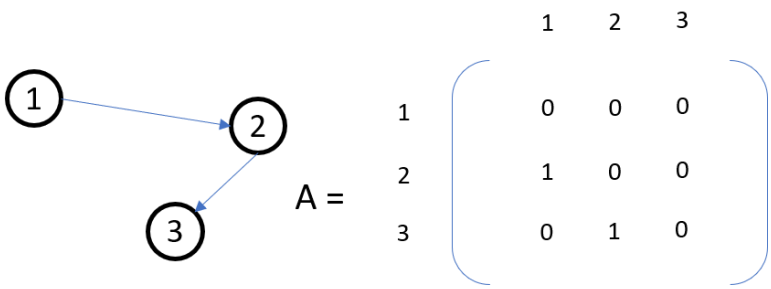
$p_1 = A * p_0 = (\ 0.375 \quad 0.083 \quad 0.333 \quad 0.208\ )^T$ (Erster Klick, Berechnungen auf Seite 2)

$p_2 = A^2 * p_0 = (\ 0.354 \quad 0.1458 \quad 0.2917 \quad 0.2083\ )^T$

$p_{50} = A^{50} * p_0 = (\ 0.387 \quad 0.129 \quad 0.290 \quad 0.194\ )^T$

Ab einem Zeitpunkt ändert sich der Zustandsvektor also nicht mehr, was der folgenden Gleichung entspricht: $\mathbf{p = A^n * p}$ Das ist eine Eigenwertgleichung zum Eigenwert 1, was der stationäre Zustand repräsentiert: $\mathbf{1 * p = A^n * p}$ Mathematisch formuliert nennt sich dieses Verhalten “Konvergenz”.

Bis anhin ist es um ein Netzwerk gegangen, das sich nicht ändert. Die Realität ist jedoch anders, Netzwerke mit ihren Verlinkungen sind dynamisch. Seiten werden gelöscht oder hinzugefügt und die Verlinkungsstruktur ändert sich ebenfalls. Es existieren zwei Zustände, die eine Konvergenz der Matrix verhindert, diese gilt es zu entdecken und speziell zu behandeln, damit schlussendlich Konvergenz erreicht werden kann, was wünschenswert ist. Bei dem ersten Zustand geht es um sogenannte **"dangling pages"** oder auch **"Sackgassen"** genannt. Zu erkennen sind diese Seiten indem die Matrix näher betrachtet wird. Gibt es eine Zeile oder Spalte (je nach Orientierung), die nur Nullen trägt, so kann davon ausgegangen werden, dass es sich hierbei um eine "Sackgasse" handelt. Weshalb? Nun, von diesem Knoten aus gibt es keine Hyperlinks zu anderen Knoten. Das Problem bei der Sache ist nun, dass der Zustandsvektor nach einer Zeit schnell null wird, sprich die Wahrscheinlichkeiten verschwinden. Dies soll an einem Beispiel illustriert werden, indem Abbild 2 etwas verändert wird, siehe dazu das Abbild 5. Die dazgehörige Adjazenzmatrix wird ebenfalls gleich mitbestimmt. Es ist nicht schwierig zu erkennen, dass bereits bei wenigen Multiplikationen die Adjazenzmatrix lauter Nullen aufweisen wird. Es gilt nun in diesem Falle alle Spalten, hier nur die dritte Spalte, aufzufinden und die Nullwerte mit 1/n auszutauschen, wobei n die Anzahl der Knoten ist, also drei. Dies soll Abbild 6 illustrieren. Wird nun wie gewohnt die Matrixmultiplikation angewendet, so entsteht daraus ein stationärer Zustand, sprich die Konvergenz ist vorhanden, das Problem der "dangling pages" ist gelöst. Es handelt sich hierbei um eine Teillösung der Google Matrix für "dangling pages" die mit dem sogenannten dyadischen Produkt berechnet wird: $\mathbf{M = A + 1/n * e \otimes a^T}$



Abbild 5 "dangling page" mit Adjazenzmatrix

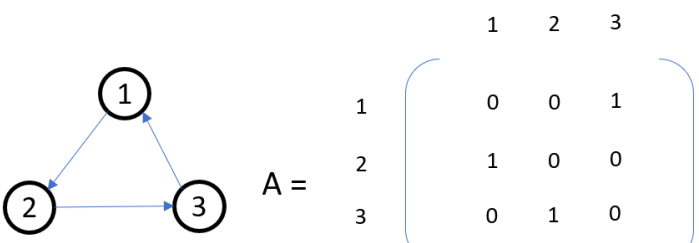
A = 1 2 3

1	0	0	1/3
2	1	0	1/3
3	0	1	1/3

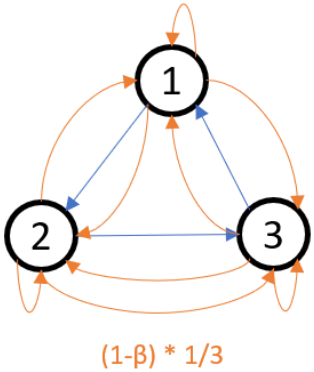
Abbild 6 korrigierte Adjazenzmatrix

A ist hierbei die Adjazenzmatrix, n die Anzahl Knoten, e der Einheitsvektor, a^T ist in diesem Fall [0 0 1], da die ersten beiden Spalten kein "dangling page" enthalten, die dritte Spalte jedoch schon. Die erhaltene Matrix M entspricht der korrigierten Adjazenzmatrix aus Abbild 6. Diese multipliziert mit dem Zustandsvektor ergibt dann die finalen korrigierten Wahrscheinlichkeit für das System.

Des Weiteren gibt es den bereits erwähnten zweiten Zustand, der zu keiner Konvergenz führt, dieser Zustand wird **„spider traps“** genannt. Hierbei entsteht bei der Multipliaktion kein Nullvektor, sondern die Matrix entwickelt sich periodisch, was bedeutet, dass sich die Einträge wiederholen. Dies wird im Abbild 7 dargestellt. Die Einträge in dieser Ausgangslage wiederholen sich nach ein paar Versuchen, was „zyklisch“ oder „periodisch“ genannt wird. Da in diesem Fall auch eine Matrix erwünscht ist, die nicht zur Periodizität führt, müssen auch hier Korrekturmassnahmen vorgenommen werden. Dazu wird ein sogenannter Dämpfungsfaktor β eingeführt. Dies wird in Abbildung 8 illustriert. All die zusätzlichen orangen Pfeile entsprechen dem Wert **(1-β)* 1 / n** , wobei n der Anzahl Knoten entspricht, sprich der Anzahl Seiten, also in diesem Fall drei. Die zugehörige Teillösung der Google Matrix für „spider traps“ wird ebenfalls mit dem dyadischen Produkt berechnet. $\mathbf{M = \beta * A + (1- \beta) * 1/n * e \otimes e^T}$



Abbild 7 "spider trap" mit Adjanzenzmatrix



Abbild 8 eingeführter Dämpfungsfaktor

Wünschenswert ist es nun diese beiden Teillösungen für „dangling pages“ und „spider traps“ in eine Google Matrix zusammenzuführen: $\mathbf{M = \beta * (A + 1/n * e \otimes a^T) + (1- \beta) * 1/n * e \otimes e^T}$

Mit dieser Lösung können nun beliebige Netzwerke betrachtet werden, die beliebige “dangling pages” und “spider traps” enthalten und somit die eigene Verlinkungsstruktur mit der Zeit variieren kann. Im nächsten Abschnitt soll nun ein Netzwerk als Beispiel verwendet und die Berechnungen an der Adjazenzmatrix praktisch umgesetzt werden.

Python-Applikation zur Berechnung von Page-Rank

Zuerst soll ein vereinfachtes Netzwerk als Berechnungsgrundlage dienen. Das vereinfachte Modell besteht aus sieben Seiten. Darin eingebaut gibt es eine „dangling page“ und ein „spider trap“.

In einem ersten Versuch soll gesehen werden, ob die dazugehörige angepasste Adjazenzmatrix (angepasst bedeutet hier in die Google Matrix überführt, wie dies bei der roten Gleichung auf Seite 5 beschrieben ist) nach einer bestimmten Anzahl Wiederholungen der Multiplikation fixe Werte aufweist oder nicht (wenn hier von „fix“ gesprochen wird, ist hier immer eine Konvergenz gemeint, die einem fixen Wert ähnelt. Im Programm selbst wird dies erreicht, indem drei Nachkommastellen als Grenze angenommen werden. Denn Konvergenz bedeutet ja immer eine Annäherung, die nur als Grenze betrachtet werden kann, wenn hier eine Grenze angenommen wird). Gemäss der vorangegangenen Theorie dürfte dies der Fall sein, da ja wie bereits erwähnt eine „dangling page“ und ein „spider trap“ eingebaut ist, die mit der roten Gleichung berücksichtigt werden. Es kann eine Konvergenz gegen einen bestimmten Wert erwartet werden, da eine behandelte Adjazenzmatrix dies auslösen kann.

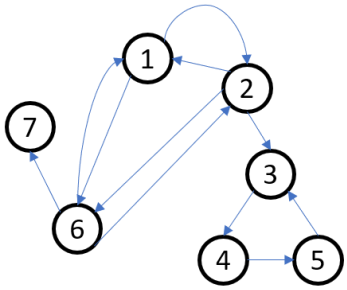
In einem zweiten Versuch soll die Berechnung an der unbehandelten Adjanzenzmatrix betrachtet werden. Auch hier soll gesehen werden, ob nach einer bestimmten Anzahl Wiederholungen der Multiplikation die Matrix fixe Werte aufweist oder nicht. Was gemäss der Erwartungen so sein sollte. Eine sich schnell entwickelte Konvergenz gegen null wird erwartet.

Es soll hier vermerkt sein, dass sich die Anzahl „Klick“ bis zur Konvergenz von Programm zu Programm unterscheiden können. Ausschlaggebend ist hier die bereits erwähnte Annahme einer eigenen Grenze. Sollte keine eigene Grenze angenommen werden, so wird bereits mit den von Hand ausgelösten Rechnungen in Octave (die Matrix potenziert und multipliziert mit dem Zustandsvektor) klar, dass je nach Anzahl Nachkommastellen das Auftreten der Konvergenz zu unterschliedlichen Potenzen, hier die Klicks, stattfinden kann. Wie nahe die Werte zu einem fixen Wert gelangen ist also Entscheidungssache, die in diesem Beispiel mit drei Nachkommastellen angenommen wird.

Es bietet sich an gleichzeitig die Erklärung des Codes mit dem eigentlichen Code im Jupyter Notebook zu lesen.

Das Netzwerk als Gegenstand der Betrachtung

Das Netzwerk soll aus sieben Seiten bestehen. Die Seite/der Knoten sieben soll eine „dangling page“ sein und die Seiten/Knoten drei, vier und fünf stellen eine „spider trap“ dar. Dieses Netzwerk soll auf Konvergenz überprüft und falls nötig dementsprechend angepasst werden, sollte die Matrix keine fixen/feste Werte annehmen und dies aufgrund des entstehenden Nullvektors bei der „dangling page“ oder der auftretenden Periodizität bei der „spider trap“. Die dazugehörige Adjazenzmatrix ist in Abbild 10 sichtbar.



Abbild 9 zu untersuchendes Netzwerk

	1	2	3	4	5	6	7
1	0	1/3	0	0	0	1/3	0
2	1/2	0	0	0	0	1/3	0
3	0	1/3	0	0	1	0	0
4	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0
6	1/2	1/3	0	0	0	0	0
7	0	0	0	0	0	1/3	0

Abbild 10 Adjazenzmatrix der sieben Knoten/ Seiten

Erster Fall mit angepasster Adjazenzmatrix

Anpassung der Adjazenzmatrix

Als erstes soll die Adjazenzmatrix festgelegt werden. Die Orientierung ist senkrecht, was bedeutet, dass beim ersten Knoten angefangen wird und die Relationen nach unten geführt werden, so wie bei dem Beispiel „Das Netzwerk als Gegenstand der Betrachtung“. Die Einträge können von Hand vorgenommen werden. Gemäss der roten Gleichung auf Seite 5 soll der erste Teil links von dem Pluszeichen berechnet werden. In diesem Abschnitt werden allfällige „dangling pages“ behandelt. Die Variable e1 trägt den Einheitsvektor, a1 repräsentiert das Vorhandensein von „dangling pages“, die mit einer eins befestigt wird. In diesem Fall haben wir alles null Einträge und am Ende einen einzigen Eintrag mit eins, da wir ja auch nur am Knoten sieben eine „dangling page“ haben. Es soll nun das erste dyadischde Produkt berechnet werden, indem np.outer verwendet wird. Dieses Ergebnis wird mit 1/7 multipliziert und zur Ausgangsmatrix addiert, das Ergebnis wird in einer Variablen „matrix“ gespeichert. Nun soll gemäss der roten Gleichung der zweite Teil rechts von dem Pluszeichen berechnet werden. Hier geht es um die Behandlung von den sogenannten „spider traps“. Mit den beiden Einheitsvektoren e2 und a2 soll das zweite dyadische Produkt gebildet werden, was in einer Variable „dyadisches_produkt2“ gespeichert wrid. Um nun die Google Matrix berechnen zu können, gilt es nun in einem letzten Schritt die berechneten Werte

mit einem Dämpfungsfaktor von 0.8 zusammenzuführen. Dies kann in Zeile 104 (Python Code) oder In [7] Zeile 3 (Jupyter Notebook) gelesen werden. Die rote Gleichung ist nun komplett berechnet. Wir haben nun die Google Matrix an der nun die Konvergenz untersucht werden kann. Alle Elemente der Google Matrix sollen auf vier Nachkommastellen gerundet werden. Dies kann ebenfalls im kommentierten Code selbst nachvollzogen werden.

Berechnung eines stationären Vektors

In einer simplen Schleife (der Schlusswert kann hier manuell abgeändert werden. In diesem Falle beträgt der Wert 20 Widerholungen, in denen eine Konvergenz stattfinden sollte) wird nun die Google Matrix potenziert und mit dem Zustandsvektor multipliziert. Das Ergebnis wird mit dem vorangegangenen Ergebnis verglichen, sollten diese dieselben sein, ist die Konvergenz erreicht und somit der stationäre Vektor bestimmt. Des Weiteren sind auch die Anzahl der Klicks bekannt da ja auch die Anzahl Wiederholungen bekannt sind. Zuerst soll der Zustandsvektor B als Vektor mit den Elementen 1/7 definiert werden. Die Google Matrix wird in eine neue Matrix A kopiert. Diese wird in der Schleife mit der Google Matrix mit dem Befehl np.dot multipliziert, das Ergebnis wird wiederum in der Variablen A festgehalten. Anschliessend multipliziert das Programm A mit B und dies ebenfalls mit dem Befehl np.dot. Das Ergebnis wird in die Variable nZ festgehalten, es handelt sich hierbei um den nächsten Zustandsvektor. Um nun überhaupt bestimmen zu können, ob eine mögliche Konvergenz vorliegt oder nicht, wird A erneut mit der Google Matrix mit dem Befehl np.dot multipliziert, was einem weiteren „Klick“ entspricht. Auch hier wird das Ergebnis wie eben in die Variable A gespeichert. In die Variable nZ2 wird das eben gerade berechnetet Produkt von A mit B gespeichert. Wir haben nun zwei Zustandsvektoren nZ und nZ2, die wir nun vergleichen können. Doch zuerst sollen diese Werte auf drei Nachkommastellen gerundet werden. Die einzelnen Elemente der Vektoren sollen nun verglichen werden, gibt es eine Übereinstimmung, hier mit eine if Anweisung und AND Verknüpfung, dann haben wir einen ersten Konvergenzfall gefunden.

In diesem Beispiel wird dies bei i = 9 erreicht. Mit der Gleichung (i * 2) + 2[1] können dazu die „Klicks“ berechnet werden, in diesem Falle ist das der Vergleich zwischen „Klick“ 20 und „Klick“ 21.

[1] Beim ersten Schleifendurchgang wird A^2 und A^3 berechnet, diese werden benötigt, um einen Vergleich anstellen zu können. So wird auch für die nächsten Schleifendurchgänge fortgegangen:
bei i= 0 wird A^2 und A^3 berechnet,
bei i = 1 ist es A^4 und A^5,
bei i = 2 ist es A^6 und A^7, aus einem beliebigen i kann nun mit dieser Gleichung die Anzahl Klicks bestimmt werden.

Zweiter Fall mit unangepasster/ unbehandelter Adjazenzmatrix

Berechnung der Zustandsvektoren

Wir wollen nun in einer weiteren Schleife die unbehandelte Adjazenzmatrix verwenden. Alle dazu berechneten Zustandsvektoren werden bis zu den „Klicks“ 21 ausgegeben, was ja dem Konvergenzfall aus dem vorangegangenen Abschnitt entspricht. Als erstes soll die Variable A erneut verwendet werden, dieses mal trägt sie die Matrix adjm, was der unbehandelten Adjazenzmatrix entspricht. Auch hier soll der Vektor B mit den Anfangswerten 1/7 bestimmt werden. Anschliessend wird die zweite Schleife betreten, die hier bis 9 iterieren wird, was aus dem vorangegangenen Abschnitt entnommen werden kann. Wie bei der ersten Schleife auch, werden zwei Zustandsvektoren berechnet. Diese werden nur gerundet und nicht verglichen, sondern direkt mit print und etwas Text ausgegeben.

Interpretation des Resultats

Gemäss der Theorie sollten sich die eingebaute «dangling page» und «spider trap» bei einer unbehandelten Adjazenzmatrix nach einer Anzahl Iterationen im Resultat des Zustandsvektors bemerkbar machen. Die erste Entwicklung wir deutlich bei den Knoten/ Seiten 1, 2, 6 und 7. Diese konvergieren rasch gegen null. Die Theorie stimmt also mit dem praktisch errechneten Resultat überein, die “dangling page” ist nun bei Knoten 6 und 7 ersichtlich, wie zu erwarten. Nun müsste noch die “spider trap” in den Wahrscheinlichkeiten ersichtlich sein. Gemäss der Theorie sollen bei den “spider traps” Periodizitäten erkannt werden, sprich mit dem Fortschreiten der Potenzierung sollen sich die Zahlenwerte, hier die Wahrscheinlichkeiten, wiederholen. Dafür soll der Zustandsvektor ab der Potenzierung 18 (kann auch früher betrachtet werden) Gegenstand der Untersuchung sein. In der Tat rotieren die Werte bei Knoten/ Seite 3,4 und 5, was der eben erwähnten Periodizität entspricht. Wir halten also bsw. den Wert 0.196 fest und können erkennen, dass dieser rotiert:

(A**18)*B	(A**19)*B	(A**20)*B	(A**21)*B
[[0.001]	[[0.001]	[[0.001]	[[0.001]
[0.001]	[0.001]	[0.001]	[0.001]
[0.196]	[0.232]	[0.214]	[0.196]
[0.214]	[0.196]	[0.232]	[0.214]
[0.231]	[0.214]	[0.196]	[0.232]
[0.001]	[0.001]	[0.001]	[0.001]
[0.001]]	[0.]]	[0.]]	[0.]]

Zusätzliche Untersuchung der Konvergenz

Es soll nun gesehen werden, ob die unangepasste Adjazenzmatrix bis zum «Klick» 21 eine Konvergenz aufweist oder nicht (zur Wiederholung: bei der angepassten Adjazenzmatrix ist zwischen dem «Klick» 20 und 21 eine Konvergenz aufgetreten). Zu erwarten ist keine Konvergenz, das ja die Periodizität der «spider trap» dies verhindert, weil die Werte stets zwischen den Knoten/ Seiten 3, 4 und 5 rotieren, wie wir das aus dem vorangegangenen Abschnitt gesehen haben. Dazu soll ln [7] Zeile 3 (Jupyter Notebook) leicht abgeändert werden: Google_Matrix = adjm (im Python Code entspricht dies Zeile 104)

Das Resultat ist eindeutig, die Konvergenz kann **nicht** erreicht werden, siehe dazu die Resultate in dem Jupyter Notebook.

Fazit

Anhand eines selbst gewählten einfachen Beispiels ist die Adjazenzmatrix gemäss der Theorie angepasst worden. Eine Konvergenz ist wie zu erwarten zwischen den „Klicks“ 20 und 21 erreicht worden:

[[0.087] Knoten 1 mit einer Wahrscheinlichkeit von 8.7 %
[0.096] Knoten 2 mit einer Wahrscheinlichkeit von 9.6 %
[0.231] Knoten 3 mit einer Wahrscheinlichkeit von 23.1 %
[0.22] Knoten 4 mit einer Wahrscheinlichkeit von 22 %
[0.212] Knoten 5 mit einer Wahrscheinlichkeit von 21.2 %
[0.096] Knoten 6 mit einer Wahrscheinlichkeit von 9.6 %
[0.061]] Knoten 7 mit einer Wahrscheinlichkeit von 6.1 %

Ebenfalls haben wir zeigen können, dass bei einer unangepassten Adjazenzmatrix keine Konvergenz auftreten kann, da ja hauptsächlich die „spider trap“ dies aufgrund den Periodizitäten verhindert. Auch der relativ schnell erreichte Nullvektor bei den „dangling pages“ kann aus dem Zustandsvektor gelesen werden, dessen Werte so nicht zu verwenden sind.

Quellen

Mathematische Grundlagen des Page Ranks, Videotutorials von Markus Geuss