Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

Projekt 3

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Thorsten Klößner, M. Sc.
Julia Wichlacz, M. Sc.
Maximilian Fickert, M. Sc.

PageRank (18 Punkte)

In diesem Projekt werden Sie anhand des PageRank-Algorithmus eine Knotengewichtung in gerichteten Graphen berechnen.

1 Der PageRank-Algorithmus

Die Idee des PageRank-Algorithmus kann an vielen Stellen im Internet nachgelesen werden, beispielsweise auf Wikipedia [1]. In diesem Projekt werden Sie nur mit dem normierten PageRank arbeiten, das heißt die Summe aller Knotengewichte ergibt immer 1.

1.1 Intuitive Erklärung: Zufallssurfer

Der PageRank-Algorithmus wurde ursprünglich für die Anwendung auf verlinkte Internetseiten entwickelt. Hierbei stellt das gesamte Internet den zugrundeliegenden Graphen dar, wobei die Knoten einzelne Internetseiten repräsentieren, und die Kanten sich aus den Links zwischen Seiten ergeben.
Intuitiv kann man den PageRank einer Seite als die Wahrscheinlichkeit verstehen, dass ein Surfer, der sich zufällig im Internet bewegt, nach einer großen Anzahl Schritte genau auf dieser Seite enden wird. Dies entspricht der relativen Anzahl an Vorkommen dieser Seite auf der unendlichen Sequenz der besuchten Seiten eines zufälligen Surfers.
Hierbei wird davon ausgegangen, dass der Zufallssurfer in jedem Schritt mit Wahrscheinlichkeit $1 - P$ einem Link auf der Seite folgt, und mit Wahrscheinlichkeit P eine beliebige zufällig ausgewählte Seite aufruft. Enthält eine Seite keine Links, ruft der Zufallssurfer immer eine zufällig ausgewählte Seite ab.
Dieses Modell lässt sich nutzen, um durch ein Zufallsexperiment den PageRank aller Seiten anzunähern. Hierzu simuliert man N Schritte des Zufallssurfers, und zählt, wie oft jede Seite besucht wurde. Der approximierte PageRank ergibt sich dann als diese Anzahl geteilt durch N.
Im nebenstehenden Beispiel könnte der Zufallssurfer beispielsweise diese Folge von 50 Seiten besuchen:
`ABDDBDABDBCDCACDBDACDACDA`
`BDDCDBDBCDBDCDDDBCDDBDBAC`
Zählt man die Vorkommen jeder Seite, so stellt man fest, dass Seite A sieben mal besucht wurde, Seite B zwölf mal, Seite C 10 mal und Seite D 21 mal. Hieraus ergeben sich folgende approximierte PageRanks:

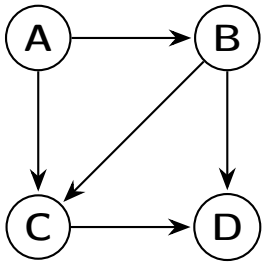

Abbildung 1: Beispiel eines Graphen mit gerichteten Kanten.

$$
\begin{aligned}
PageRank_{random,50}(A) &= 7/50 = 0.14 \\
PageRank_{random,50}(B) &= 12/50 = 0.24 \\
PageRank_{random,50}(C) &= 10/50 = 0.20 \\
PageRank_{random,50}(D) &= 21/50 = 0.42
\end{aligned}
$$

1.2 Mathematische Berechnung

Das Modell des Zufallssurfers lässt sich auch als Markow-Kette erster Ordnung darstellen. Hierzu werden Übergangswahrscheinlichkeiten paarweise zwischen allen Zuständen definiert. Beispielsweise ist die Übergangswahrscheinlichkeit von Knoten B zu Knoten D bei $P = 0.1$ die Summe aus $\frac{0.1}{4}$ (Wahrscheinlichkeit, durch zufällige Auswahl zu Knoten D zu gelangen) und $\frac{0.9}{2}$ (Wahrscheinlichkeit, die ausgehende Kante von B zu wählen).

Auf diese Weise lassen sich alle Übergangswahrscheinlichkeiten berechnen:

$$\begin{aligned} P_{X,Y} &= \begin{cases} \frac{1}{N} & \text{falls } out(X) = 0 \\ \frac{P}{N} + \frac{(1-P)*out(X,Y)}{out(X)} & \text{sonst.} \end{cases} \\ out(X) &= \sum_{Y} out(X,Y) \end{aligned}$$

Hierbei bezeichnet N die Anzahl der Knoten im Graphen, $out(X,Y)$ die Anzahl der Kanten von X nach Y (von denen es beliebig viele geben kann) und $out(X)$ die Gesamtanzahl ausgehender Kanten von X. $P_{X,Y}$ ist die Wahrscheinlichkeit, von Knoten X nach Y zu gelangen.
Für das Beispiel aus Abbildung 1 und $P = 0.1$ würde sich dadurch die folgende Übergangsmatrix ergeben:

$$M = \begin{bmatrix} 0.025 & 0.475 & 0.475 & 0.025 \\ 0.025 & 0.025 & 0.475 & 0.475 \\ 0.025 & 0.025 & 0.025 & 0.925 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

Diese kodiert in Zeile i, Spalte j jeweils die Übergangswahrscheinlichkeit von Knoten i zu Knoten j (in diesem Fall alphabetisch sortiert). Sie kann nun ebenfalls dazu verwendet werden, den PageRank einer Seite anzunähern. Hierzu multipliziert man die Matrix wiederholt mit einem Wahrscheinlichkeitsvektor V, der die aktuelle Wahrscheinlichkeit kodiert, mit der man sich in jedem Knoten aufhält. Diesen Vektor initialisiert man mit $\frac{1}{N}$ für jeden Knoten. In jedem Schritt ergibt sich der neue Vektor nun aus der Multiplikation $V * M$:

$$\begin{aligned} V_0 &= \left[\frac{1}{N} \cdots \frac{1}{N}\right] \\ V_i &= V_{i-1} * M \quad \text{für } i > 0 \end{aligned}$$

Das heißt also, die Wahrscheinlichkeit sich in Schritt s an einem bestimmten Knoten X zu befinden ergibt sich aus der Summe der Wahrscheinlichkeiten vorher an einem Knoten Y zu sein jeweils multipliziert mit der Wahrscheinlichkeit sich von Y zu X zu bewegen. Für den Knoten j in Schritt s ergibt sich das also aus:

$$v_{s,j} = \sum_{i=1}^{N} v_{s-1,i} * m_{i,j}$$

Dabei bezeichnet $m_{i,j}$ den Eintrag in Zeile i und Spalte j der Übergangsmatrix M und $v_{s,i}$ den i-ten Eintrag des Wahrscheinlichkeitsvektors V_s in Schritt s.
Für das Beispiel aus Abbildung 1 ergeben sich folgende Wahrscheinlichkeitsvektoren:

$$\begin{aligned} V_0 &= \begin{bmatrix} 0.25000 & 0.25000 & 0.25000 & 0.25000 \end{bmatrix} \\ V_1 &= \begin{bmatrix} 0.08125 & 0.19375 & 0.30625 & 0.41875 \end{bmatrix} \\ V_2 &= \begin{bmatrix} 0.11921 & 0.15578 & 0.24296 & 0.48203 \end{bmatrix} \\ V_3 &= \begin{bmatrix} 0.13345 & 0.18710 & 0.25720 & 0.42223 \end{bmatrix} \\ V_4 &= \begin{bmatrix} 0.12000 & 0.18005 & 0.26425 & 0.43568 \end{bmatrix} \\ &\cdots \\ V_{10} &= \begin{bmatrix} 0.12346 & 0.17908 & 0.25969 & 0.43775 \end{bmatrix} \\ &\cdots \\ V_{20} &= \begin{bmatrix} 0.12349 & 0.17907 & 0.25965 & 0.43777 \end{bmatrix} \end{aligned}$$

Diese Wahrscheinlichkeitsvektoren konvergieren gegen den tatsächlichen PageRank, sogar unabhängig vom gewählten Initialvektor (solange die Summe der Einträge 1 ergibt). Diesen Grenzwert nennt man die stationäre Verteilung der Markow-Kette.

2 Aufgaben

Ihre Aufgabe in diesem Projekt ist es, sowohl den Zufallssurfer als auch die Berechnung mittels Markow-Kette zu implementieren. Dieses Kapitel beschreibt detailliert, welche Kriterien ihre Implementierung zu erfüllen hat.

2.1 Format der Abgabe

Sämtliche Teile ihrer Implementierung haben in der Programmiersprache C zu erfolgen. Sie dürfen den neuesten Sprachstandard (C11) benutzen, jedoch keine externen Bibliotheken. Die Standardbibliotheken (`stdio.h`, `stdlib.h`, `string.h`, ...) dürfen Sie natürlich benutzen. Ihr Projekt muss in der zur Verfügung gestellten virtuellen Maschine durch den Aufruf von *make* eine ausführbare Binärdatei mit dem Namen *pagerank* erzeugen.

2.2 Eingabeformat

Die Graphen, die ihr Programm verarbeitet, haben eine fest vorgegebene Form. Jede Datei beginnt mit dem Schlüsselwort „digraph“, gefolgt von einem Bezeichner für den Graphen und einer öffnenden geschweiften Klammer. Anschließend wird pro Zeile eine Kante im Graphen definiert, gefolgt von einer letzten Zeile, die nur eine schließende geschweifte Klammer enthält. Für jede Kante wird zunächst der Bezeichner des ausgehenden Knotens angegeben, dann die Zeichenfolge „ `->` “, der Bezeichner des erreichten Knotens, und ein abschließendes Semikolon. Die Zeichenfolge zwischen den Bezeichnern besteht genau aus einem Leerzeichen, einem Pfeil bestehend aus Minus-Zeichen und Größer-Als-Zeichen, und einem weiteren Leerzeichen. Zeilenwechsel werden jeweils durch ein Newline-Zeichen („`\n`“) kodiert. Jeder Bezeichner beginnt mit einem Buchstaben, möglicherweise gefolgt von weiteren Buchstaben oder Ziffern. Bezeichner bis zu einer Länge von 256 Zeichen müssen unterstützt werden. Der Beispielgraph aus Abbildung 1 würde wie folgt kodiert:

```
digraph TestGraph {
A -> B;
A -> C;
B -> C;
B -> D;
C -> D;
}
```

Es ist ausreichend, wenn ihr Programm nur genau diese Repräsentation verarbeiten kann. Sie dürfen jedoch beispielsweise bei der Setzung der Leerzeichen und Zeilenwechsel auch weniger restriktiv sein, also mehr akzeptieren als hier spezifiziert.
Graphen in diesem Format können auch komfortabel mit den Programmen des Graphviz-Paketes [2] visualisiert werden. Wenn Sie die Graphviz Programme in der VM benutzen möchten können Sie das Paket mit folgenden Befehlen installieren:

```
sudo pacman -Syu
sudo pacman -Sy graphviz
```

2.3 Ausgabeformat

Der berechnete PageRank wird jeweils auf der Standardausgabe in einer Zeile pro Knoten ausgegeben, wobei zunächst der Bezeichner des Knotens ausgegeben wird, dann mindestens ein Leerzeichen oder Tabulatorzeichen, und dann der PageRank mit einer Präzision von mindestens 10 Nachkommastellen.
Die Reihenfolge, in der die Knoten ausgegeben werden, spielt keine Rolle.
Der PageRank für den Graphen aus Abbildung 1 könnte also so aussehen:

```
A       0.1234987158
B       0.1790731292
C       0.2596560461
D       0.4377721089
```

2.3.1 Statistikausgabe

Unter gewissen Umständen (siehe Abschnitt 2.4) soll kein PageRank berechnet, sondern lediglich eine Statistik über den Graphen ausgegeben werden. Hierbei werden die folgenden Informationen ausgegeben:

- Bezeichner des Graphen
- Anzahl der Knoten im Graphen
- Anzahl der Kanten im Graphen
- Minimaler und maximaler Eingangsgrad (=Anzahl eingehender Kanten)
- Minimaler und maximaler Ausgangsgrad (=Anzahl ausgehender Kanten)

Das genaue Ausgabeformat können Sie aus dieser Beispielausgabe des Graphen aus Abbildung 1 entnehmen:

```
TestGraph:
- num nodes: 4
- num edges: 5
- indegree: 0-2
- outdegree: 0-2
```

2.4 Kommandozeilenparameter

Ihr Programm muss die folgenden Kommandozeilenparameter entgegennehmen und verarbeiten können:

-h Eine kurze Hilfe zu den vorhandenen Kommandozeilenparametern wird ausgegeben.

-m N Es wird die Berechnung auf Grundlage der Markow-Kette mit **N** Schritten vorgenommen, und das Ergebnis ausgegeben. **N** ist eine ganze Zahl größer 0.

-p N Der Parameteter P für die Berechnung des PageRank wird auf **N**% festgelegt. **N** ist eine ganze Zahl zwischen 1 und 100. Wird dieser Parameter nicht spezifiziert, ist $P = 10\%$.

-r N Es wird eine Simulation des Zufallssurfers mit **N** Schritten vorgenommen, und das Ergebnis ausgegeben. **N** ist eine ganze Zahl größer 0.

-s Es wird eine Statistik des Graphen ausgegeben, wie in 2.3.1 spezifiziert.

Es muss mindestens einer der Parameter **-h**, **-m**, **-r** oder **-s** angegeben sein[1].
Wenn nicht der Parameter **-h** angegeben wurde, wird nach den beschriebenen Optionen genau ein weiteres Argument erwartet. Dieses gibt den Dateinamen des Eingabegraphen an. Er kann relativ zum aktuellen Arbeitsverzeichnis oder als absoluter Pfad angegeben werden, beginnt jedoch nie mit einem Minus.

2.5 Details der PageRank-Implementierung

Um ein einheitliches Verhalten aller Projekt-Implementierungen zu gewährleisten, spezifizieren wir hier einige Details der PageRank-Implementierung. Diese ergeben sich nicht aus der allgemeinen Definition aus Abschnitt 1.

1. Wie bereits erwähnt, soll die Ausgabe des PageRank immer so normiert sein, dass die Summe 1 ergibt.
2. Eine Simulation des Zufallssurfers mit N Schritten bedeutet, dass initial ein zufälliger Knoten gewählt wird, und anschließend N Schritte ausgeführt werden, wobei die dabei erreichten Knoten aufgezeichnet und gezählt werden. Der initiale Knoten zählt dabei nicht mit.
3. Eine Berechnung auf der Markow-Kette mit N Schritten auszuführen bedeutet, einen Wahrscheinlichkeitsvektor mit gleichverteilten Wahrscheinlichkeiten zu initialisieren, und anschließend N mal mit der Matrix zu multiplizieren.

   Hierbei ist es unerheblich, ob Sie die Matrix tatsächlich im Speicher darstellen und eine Matrixmultiplikation durchführen, oder die Berechnung direkt auf der Basis des Graphen ausführen. Das Endergebnis muss natürlich das gleiche sein.

[1] Wie sich das Programm verhält, wenn mehr als einer dieser Parameter angegeben ist, ist nicht spezifiziert. Sie dürfen jedes Ihnen sinnvoll erscheinende Verhalten implementieren.

2.6 Programmbeendigung (2 Bonuspunkte)

Im regulären Fall soll das Programm immer mit Exit-Code 0 beendet werden. Im Fehlerfall wird immer mit 1 beendet.
Sämtliche Fehlerbehandlung ist in diesem Projekt optional. Sie dürfen sich darauf verlassen, dass ihr Programm nur mit gültigen Kommandozeilenparametern aufgerufen wird, und die Eingabedatei das korrekte Format hat.
Wenn Sie Fehlererkennung implementieren, können Sie dadurch bis zu 2 Bonuspunkte sammeln. Folgende Fehler können beispielsweise auftreten:

- Es werden ungültige Kommandozeilenparameter übergeben.
- Die Eingabedatei existiert nicht.
- Die Eingabedatei hat ein ungültiges Format.

Die Tests für den Bonusteil werden zusammen mit den geheimen Tests nur einmal nach Abgabe des Projekts ausgeführt. Dabei wird lediglich überprüft ob ihr Programm in diesen Fällen mit Exit-Code 1 beendet wird.

3 Vorschlag zur Vorgehensweise

Dieses Projekt stellt bereits ein mittelgroßes, nicht-triviales Softwareprojekt dar. Insbesondere wenn Sie noch nicht viel Erfahrung in der Softwareentwicklung haben, empfiehlt es sich, die einzelnen Teile des Gesamtprojektes nacheinander zu implementieren und zu testen.
Um Sie beim Zeitmanagement zu unterstützen geben wir Ihnen Meilensteine vor, zu denen Sie bestimmte Teile des Projekts implementiert haben sollten. Das Einhalten der Meilensteine ist nicht verpflichtend, sondern dient ihnen hauptsächlich dazu den Zeitaufwand der einzelnen Teile des Projekts besser einzuschätzen, sodass Sie sich die Zeit besser einteilen können und einen Überblick über Ihren Fortschritt haben.
Kompilieren Sie ihren Code regelmäßig und überprüfen Sie, dass er funktioniert. Vergessen Sie außerdem nicht, Zwischenstände regelmäßig in der Versionsverwaltung zu archivieren, und damit zu sichern.

1. Klonen Sie das Depot in dem sie den Befehl

   ```
   git clone https://prog2scm.cdl.uni-saarland.de/git/project3/$NAME
   ```

   wobei `$NAME` durch Ihren Benutzernamen zu ersetzen ist. Geben Sie an, dass der Klon in ein Verzeichnis namens `project3` abgelegt wird. Außerdem möchten wir Sie drauf hinweisen, dass unser Server nur aus dem Universitätsnetz erreichbar ist. Stellen Sie außerdem sicher, dass das Projekt durch den Aufruf von `make` erfolgreich kompiliert.

2. Implementieren Sie das Auslesen der Kommandozeilenparameter. Hierzu können Sie die `getopt`-Funktion [3] verwenden. Sie können sich am Beispiel in der Dokumentation orientieren, und dürfen diesen Beispielcode auch in ihr Projekt übernehmen und anpassen.

3. Überlegen Sie sich eine sinnvolle Darstellung des Graphen. Beachten Sie, welche Informationen während des Einlesens und während der PageRank-Berechnung einfach zugänglich sein müssen. Sie dürfen dafür auch bekannte Datenstrukturen verwenden, z.B. Adjazenzlisten (`https://de.wikipedia.org/wiki/Adjazenzliste`)

   Implementieren Sie nun diese Graphdarstellung. Denken Sie daran den Code regelmäßig zu kompilieren und zu testen.

4. Nachdem Sie nun Graphen im Arbeitsspeicher darstellen können, kümmern Sie sich um das Einlesen (Parsen) des Graphen aus der Eingabedatei. Hierbei ist Dynamische Speicherverwaltung sehr nützlich (siehe Skript Absatz 3.11). Ignorieren Sie zunächst sämtliche Fehler, die mit der Eingabedatei auftreten können (siehe Abschnitt 2.6).

   Ein sehr einfacher Parser lässt sich mithilfe der `fscanf`-Funktion [4] erstellen.

   Beheben Sie in diesem Schritt auch auftretende Fehler in Ihrer Graph-Implementierung.

5. Um die Korrektheit der vorherigen Schritte zu überprüfen, implementieren Sie die Kommandozeilenoption `-s` (siehe Abschnitt 2.3.1).

   Überprüfen Sie, dass für verschiedene Graphen die richtige Statistik ausgegeben wird.

   – Diesen Zwischenstand sollten Sie bis zum 30.5. erreicht haben. –

6. Beginnen Sie nun mit der Implementierung des PageRank-Algorithmus nach der Zufallssurfer-Methode. Orientieren sie sich hierfür an der Beschreibung in Abschnitt 1.1. Für die Implementierung ist es insbesondere notwendig, eine gleichverteilte Zufallszahl im Bereich $[0..X]$ erzeugen zu können. Stellen Sie sicher, dass ihr Zufall nichtdeterministisch ist, dass sich also bei jedem Aufruf andere Zufallszahlen ergeben. Dafür finden Sie in ihrem initialen Depot in der Datei `utils.c` bereits eine Implementierung, die diesen Anforderungen genügt.

7. Implementieren Sie außerdem das Verarbeiten des Kommandozeilenargumentes `-p`, der den Parameter P bestimmt.

8. Testen Sie ihre Implementierung mit verschiedenen Werten für P auf verschiedenen Graphen. Beachten Sie dabei auch Sonderfälle wie leere Graphen, nicht verbundene Graphen, Knoten ohne Nachfolger und Kanten, deren Zielknoten gleich dem Ausgangsknoten ist.

   – Diesen Zwischenstand sollten Sie bis zum 3.6. erreicht haben. –

9. Implementieren Sie nun die Berechnung auf Basis der Markow-Kette. Überprüfen Sie, dass die Berechnung gegen die stationäre Verteilung der Markow-Kette konvergiert, und die Summe der PageRanks immer 1 ergibt.

4 Tests

In ihrem Depot finden Sie einige öffentliche Tests, die Sie mit einem Aufruf von `make tests` oder alternativ `./tests/run-tests.py pagerank` ausführen können. Falls sie eine detailiertere Ausgabe wünschen können sie den Aufruf durch die Parameter `-v` oder `-d` ergänzen, z.B. `./tests/run-tests.py pagerank -v`
Bevor Sie die Tests ausführen können müssen Sie außerdem noch mit folgenden beiden Befehlen das scipy Paket für Python nachinstallieren:

```
sudo pacman -Syu
sudo pacman -Sy python-scipy glibc valgrind
```

Die public Tests werden außerdem in regelmäßigen Abständen zusammen mit geheimen *daily* Tests auf unserem Testserver ausgeführt wenn Sie neue Versionen des Projekts pushen. Sie erhalten über den Ausgang der Tests eine Benachrichtigung per Email. Nach Ende des Projekts wird Ihre Abgabe mithilfe weiterer *secret* Tests ausgewertet.

5 Abgabe

Die Abgabe erfolgt durch das Einspielen in das zur Verfügung gestellte Versionsverwaltungssystem. Die Deadline für die finale Abgabe ist der *8. Juni, 23:59 Uhr*. In die Bewertung fließen außer den öffentlichen Tests auch die daily und secret Tests ein. Zur finalen Bewertung ist ausschließlich der letzte im Versionsverwaltungssystem abgelegte Stand Ihres Codes entscheidend. Denken Sie daran Ihr Projekt nicht nur am Ende, sondern regelmäßig zu pushen damit ihr Projekt auf unseren Servern gesichert ist und Sie Feedback von den daily Tests erhalten.

Literatur

[1] `http://de.wikipedia.org/wiki/PageRank`
[2] `http://de.wikipedia.org/wiki/Graphviz`
[3] `http://www.gnu.org/software/libc/manual/html_node/Getopt.html`
[4] `http://en.cppreference.com/w/c/io/fscanf`